



**APPLICATION
NOTE**

AP-223

October 1984

**8051 Based CRT Terminal
Controller**

Michael A. Steier
Microcontroller Applications

1.0 INTRODUCTION

This is the third application note that Intel has produced on CRT terminal controllers. The first Ap Note (ref. 1), written in 1977, used the 8080 as the CPU and required 41 packages including 11 LSI devices. In 1979, another application note (ref. 2) using the 8085 as the controller was produced and the chip count decreased to 20 with 11 LSI devices.

Advancing technology has integrated a complete system onto a single device that contains a CPU, program memory, data memory, serial communication, interrupt controller, and I/O. These "computer-on-a-chip" devices are known as microcontrollers. Intel's MCS®-51 microcontroller was chosen for this application because of its highly integrated functions. This CRT terminal design uses 12 packages with only 4 LSI devices.

This application note has been divided into five general sections:

- 1) CRT Terminal Basics
- 2) 8051 Description
- 3) 8276 Description
- 4) Design Background
- 5) System Description

2.0 CRT TERMINAL BASICS

A terminal provides a means for humans to communicate with a computer. Terminals may be as simple as a LED display and a couple of push buttons, or it may be an elaborate graphics system that contains a full function keyboard with user programmable keys, color CRT and several processors controlling its functions. This application note describes a basic low cost terminal containing a black and white CRT display, full function keyboard and a serial interface.

2.1 CRT Description

A raster scan CRT displays its images by generating a series of lines (raster) across the face of the tube. The electron beam usually starts at the top left hand corner moves left to right, back to the left of the screen, moves down one row and continues on to the right. This is repeated until the lower right hand of the screen is reached. Then the beam returns to the top left hand corner and refreshes the screen. The beam forms a zigzag pattern as shown in Figure 2.1.0.

Two independent operating circuits control this movement across the screen. The horizontal oscillator controls the left to right motion of the beam while the vertical controls the top to bottom movement. The vertical oscillator also tells the beam when to return to the upper left hand corner or "home" position.

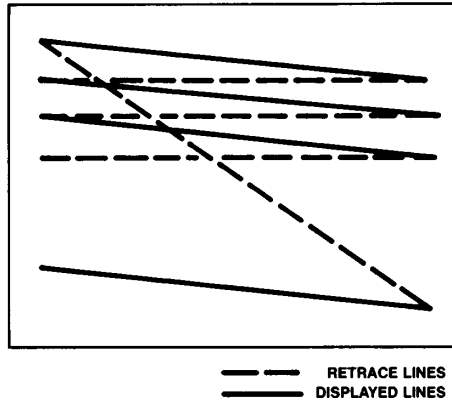


Figure 2.1.0 Raster Scan

As the electron beam moves across the screen under the control of the horizontal oscillator, a third circuit controls the current entering the electron gun. By varying the current, the image may be made as bright or as dim as the user desires. This control is also used to turn the beam off or "blank the screen".

When the beam reaches the right hand side of the screen, the beam is blanked so it does not appear on the screen as it returns to the left side. This "retrace" of the beam is at a much faster rate than it traveled across the screen to generate the image.

The time it takes to scan the whole screen and return to the home position is referred to as a "frame". In the United States, commercial television broadcast uses a horizontal sweep frequency of 15,750Hz which calculates out to 63.5 microseconds per line. The frame time is equal to 16.67 milliseconds or 60Hz vertical sweep frequency.

Although this is the commercial standard, many CRT displays operate from 18KHz to 30KHz horizontal frequency. As the horizontal frequency increases, the number of lines per frame increases. This increase in lines or resolution is needed for graphic displays and on special text editors that display many more lines of text than the standard 24 or 25 character lines.

Since the United States operates on a 60Hz A.C. power line frequency, most CRT monitors use 60Hz as the vertical frequency. The use of 60Hz as the vertical frequency allows the magnetic and electrical variations that can modulate the electron beam to be synchronized with the display, thus they go unnoticed. If a frequency other than 60Hz is used, special shielding and power supply regu-

lating is usually required. Very few CRTs operate on a vertical frequency other than 60Hz due to the increase in the overall system cost.

The CRT controller must generate the pulses that define the horizontal and vertical timings. On most raster scan CRTs the horizontal frequency may vary as much as 500Hz without any noticeable effect on the quality of the display. This variation can change the number of horizontal lines from 256 to 270 per frame.

The CRT controller must also shift out the information to be displayed serially to the circuit that controls the electron beam's intensity as it scans across the screen. The circuits that control the timing associated with the shifting of the information are known as the dot clock and the character clock. The character clock frequency is equal to the dot clock frequency divided by the number of dots it takes to form a character in the horizontal axis. The dot clock frequency is calculated by the following equation:

$$\text{Dot Clock (Hz)} = (N + R) * D * L * F$$

where

- N is the number of displayed characters per row,
- R is the number of character times for the retrace,
- D is the number of dots per character in the horizontal axis,
- L is the number of horizontal lines per frame,
- F is the frame rate in Hz.

In this design N = 80, R = 20, D = 7, L = 270, and F = 60Hz. Plugging in the numbers results in a dot clock frequency of 11.34MHz.

The retrace number may vary on each design because it is used to set the left and right hand margins on the CRT. The number of dots per character is chosen by the designer to meet the system needs. In this design, a 5 × 7 dot matrix and 2 blank dots between each character (see Figure 2.1.1) makes D equal to 5 + 2 = 7.

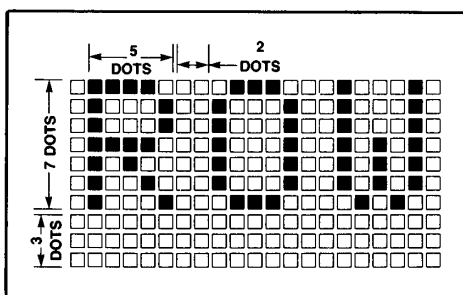


Figure 2.1.1 5 × 7 Dot Matrix

The following equation can be used to figure the number of lines per frame:

$$L = (H * Z) + V$$

where

- H is the number of horizontal lines per character,
- Z is the number of character lines per frame,
- V is the number of horizontal line times during the vertical retrace

In this design H is equal to the 7 horizontal dots per character plus 3 blank dots between each row which adds up to 10. Also 25 lines of characters are displayed, so Z = 25. The vertical retrace time is variable to set the top and bottom margins on the CRT and in this design is equal to 20. Plugging in the numbers gives L = 270 lines per frame.

2.2 Keyboard

A keyboard is the common way a human enters commands and data to a computer. A keyboard consists of a matrix of switches that are scanned every couple of milliseconds by a keyboard controller to determine if one of the keys has been pressed. Since the keyboard is made up of mechanical switches that tend to bounce or "make and break" contact everytime they are pressed, debouncing of the switches must also be a function of the keyboard controller. There are dedicated keyboard controllers available that do everything from scanning the keyboard, debouncing the keys, decoding the ASCII code for that key closure to flagging the CPU that a valid key has been depressed. The keyboard controller may present the information to the CPU in parallel form or in a serial data stream.

This Application Note integrates the function of the keyboard controller into the 8051 which is also the terminal controller. Provisions have been made to interface the 8051 to a keyboard that uses a dedicated keyboard controller. The 8051 can accept data from the keyboard controller in either parallel or serial format.

2.3 Serial Communications

Communication between a host computer and the CRT terminal can be in either parallel or serial data format. Parallel data transmission is needed in high end graphic terminals where great amounts of information must be transferred.

One can rarely type faster than 120 words per minute, which corresponds to 12 characters per second or 1 character per 83 milliseconds. The utilization of a parallel port cannot justify the cost associated with the drivers and the amount of wire needed to perform this transmission. Full duplex serial data transmission requires 3 wires and two

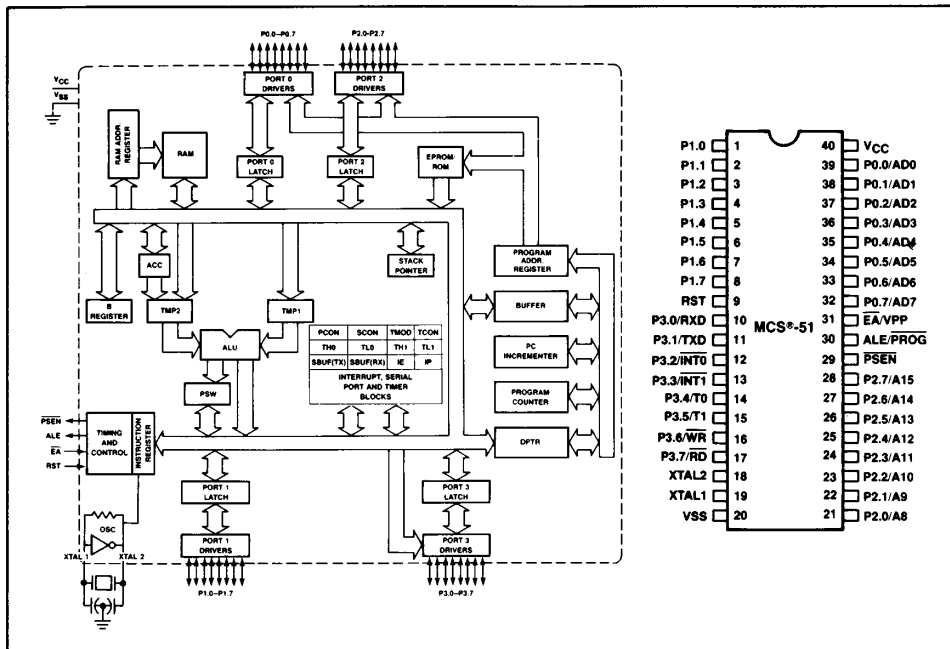


Figure 3.0.0 8051 Block Diagram

drivers to implement the communication channel between the host computer and the terminal. The data rate can be as high as 19200 BAUD in the asynchronous serial format. BAUD rate is the number of bits per second received or transmitted. In the asynchronous serial format, 10 bits of information is required to transmit one character. One character per 500 microseconds or 1,920 characters per second would then be transmitted using 19.2 KBAUD.

This application note uses the 8051 serial port configured for full duplex asynchronous serial data transmission. The software for the 8051 has been written to support variable BAUD rates from 150 BAUD up to 9.6 KBAUD.

3.0 8051 DESCRIPTION

The 8051 is a single chip high-performance microcontroller. A block diagram is shown in figure 3.0.0. The 8051 combines CPU; Boolean processor; $4K \times 8$ ROM; 128×8 RAM; 32 I/O lines; two 16-bit timer/ event counters; a five-source, two-priority-level, nested interrupt structure; serial I/O port for either multiprocessor communications, I/O expansion, or full duplex UART; and on-chip oscillator and clock circuits.

3.1 CPU

Efficient use of program memory results from an instruction set consisting of 49 single-byte, 45 two-byte and 17 three-byte instructions. Most arithmetic, logical and branching operations can be performed using an instruction that appends either a short address or a long address. For example, branches may use either an offset that is relative to the program counter which takes two bytes or a direct 16-bit address which takes three bytes to perform. As a result, 64 instructions operate in one machine cycle, 45 in two machine cycles, and the multiply and divide instruction execute in 4 machine cycles.

The 8051 has five addressing modes for source operands: Register, Direct, Register-Indirect, Immediate, and Based-Register-plus Index-Register-Indirect Addressing.

The Boolean Processor can be thought of as a separate one-bit CPU. It has its own accumulator (the carry bit), instruction set for data moves, logic, and control transfer, and its own bit addressable RAM and I/O. The bit-manipulating instructions provide optimum code and speed efficiency for handling on chip peripherals. The

Boolean processor also provides a straight forward means of converting logic equations directly into software. Complex combinational logic functions can be resolved without extensive data movement, byte masking, and test-and-branch trees.

3.2 On-Chip Ram

The CPU manipulates operands in four memory spaces. These are the 64K-byte Program Memory, 64K-byte External Data Memory, 128-byte Internal Data Memory, and 128-byte Special Function Registers (SFRs). Four Register Banks (each with 8 registers), 128 addressable bits, and the Stack reside in the internal Data RAM. The Stack size is limited only by the available Internal Data RAM and its location is determined by the 8-bit Stack Pointer. All registers except for the Program Counter and the four 8-Register Banks reside in the SFR address space. These memory mapped registers include arithmetic registers, pointers, I/O ports, and registers for the interrupt system, timers, and serial channel.

Registers in the four 8-Register Banks can be addressed by Register, Direct, or Register-Indirect Addressing modes. The 128 bytes of internal Data Memory can be addressed by Direct or Register-Indirect modes while the SFRs are only addressed directly.

3.3 I/O Ports

The 8051 has instructions that can treat the 32 I/O lines as 32 individually addressable bits or as 4 parallel 8-bit ports addressable as Ports 0, 1, 2, and 3.

Resetting the 8051 writes a logical 1 to each pin on port 0 which places the output drivers into a high-impedance mode. Writing a logical 0 to a pin forces the pin to ground and sinks current. Re-writing the pin high will place the pin in either an open drain output or high-impedance input mode.

Ports 1, 2, and 3 are known as quasi-bidirectional I/O pins. Resetting the device writes a logical one to each pin. Writing a logical 0 to the pin will force the pin to ground and sink current. Re-writing the pin high will place the pin in an output mode with a weak depletion pullup FET or in the input mode. The weak pullup FET is easily overcome by a TTL output.

Ports 0 and 2 can also be used for off-chip peripheral expansion. Port 0 provides a multiplexed low-order address and data bus while Port 2 contains the high-order address when using external Program Memory or more than 256 byte external Data Memory.

Port 3 pins can also be used to provide external interrupt request inputs, event counter inputs, the serial port TXD

and RXD pins and to generate control signals used for writing and reading external peripherals.

3.4 Interrupt System

External events and the real-time-driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. A five-source, two-level, nested interrupt system ties the real time events to the normal program execution.

The 8051 has two external interrupt sources, one interrupt from each of the two timer/counters, and an interrupt from the serial port. Each interrupt vectors the program execution to its own unique memory location for servicing the interrupt. In addition, each of the five sources can be individually enabled or disabled as well as assigned to one of the two interrupt priority levels available on the 8051.

Up to two additional external interrupts can be created by configuring a timer/counter to the event counter mode. In this mode the timer/counter increments on command by either the T0 or T1 pin. An interrupt is generated when the timer/counter overflows. Thus if the timer/counter is loaded with the maximum count, the next high-to-low transition of the event counter input will cause an interrupt to be generated.

3.5 Serial Port

The 8051's serial port is useful for linking peripheral devices as well as multiple 8051s through standard asynchronous protocols with full duplex operation. The serial port also has a synchronous mode for expansion of I/O lines using shift registers. This hardware serial port saves ROM code and permits a much higher transmission rate than could be achieved through software. The processor merely needs to read or write the serial buffer in response to an interrupt. The receiver is double buffered to eliminate the possibility of overrun if the processor failed to read the buffer before the beginning of the next frame.

The full duplex asynchronous serial port provides the means of communication with standard UART devices such as CRT terminals and printers.

The reader should refer to the microcontroller handbook for a complete discussion of the 8051 and its various modes of operation.

4.0 8276 DESCRIPTION

The 8276's block diagram and pin configuration are shown in Figure 4.0.0. The following sections describe the general capabilities of the 8276.

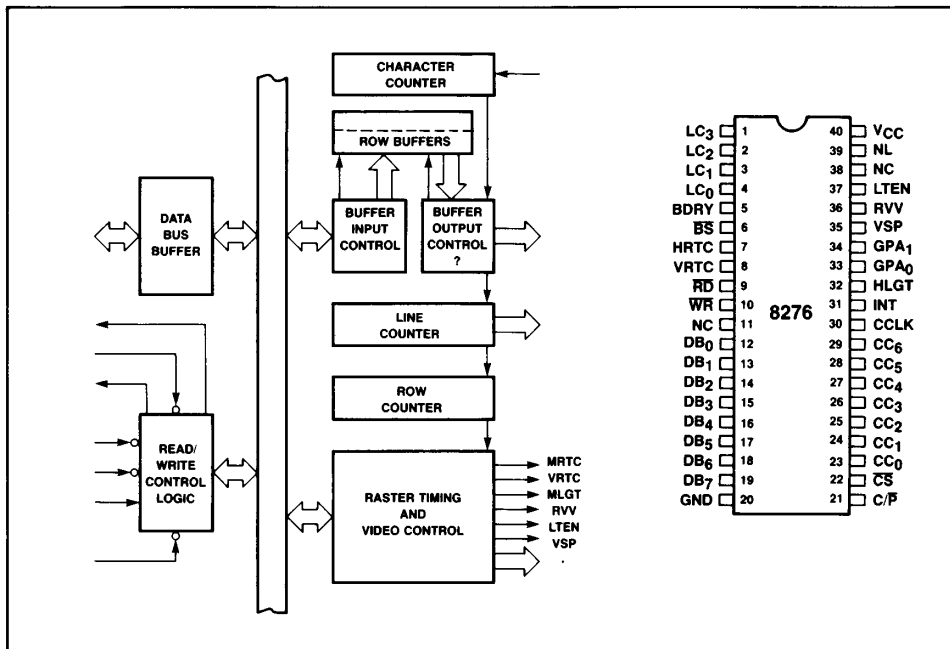


Figure 4.0.0 8276 Block Diagram

4.1 CRT Display Refreshing

The 8276, having been programmed by the system designer for a specific screen format, generates a series of Buffer Ready signals. A row of characters is then transferred by the system controller from the display memory to the 8276's row buffers. The row buffers are filled by deselecting the 8276 \overline{CS} and asserting the \overline{BS} and \overline{WR} signals. The 8276 presents the character codes to an external character generator ROM by using outputs $CC0$ – $CC6$. The parallel data from the outputs of the character generator is converted to serial information that is clocked by external dot timing logic into the video input of the CRT.

The character rows are displayed on the CRT one line at a time. Line count outputs $LC0$ – $LC3$ select the current line information from the character generator ROM. The display process is illustrated in Figure 4.1.0. This process is repeated for each display character row. At the beginning of the last display row the 8276 generates an interrupt request by raising its \overline{INT} output line. The interrupt request

is used by the 8051 system controller to reinitialize its load buffer pointers for the next display refresh cycle.

Proper CRT refreshing requires that certain 8276 parameters be programmed at system initialization time. The 8276 has two types of internal registers; the write only Command (CREG) and Parameter (PREG) Registers, and the read only Status Register (SREG). The 8276 expects to receive a command followed by 0 to 4 parameter bytes depending on the command. A summary of the 8276's instruction set is shown in Figure 4.1.1. To access the registers, \overline{CS} must be asserted along with \overline{WR} or \overline{RD} . The status of the C/P pin determines whether the command or parameter registers are selected.

The 8276 allows the designer flexibility in the display format. The display may be from 1 to 80 characters per row, 1 to 64 rows per screen, and 1 to 16 horizontal lines per character row. In addition, four cursor formats are available; blinking, non-blinking, underline, and reverse video. The cursor position is programmable to anywhere on the screen via the Load Cursor command.

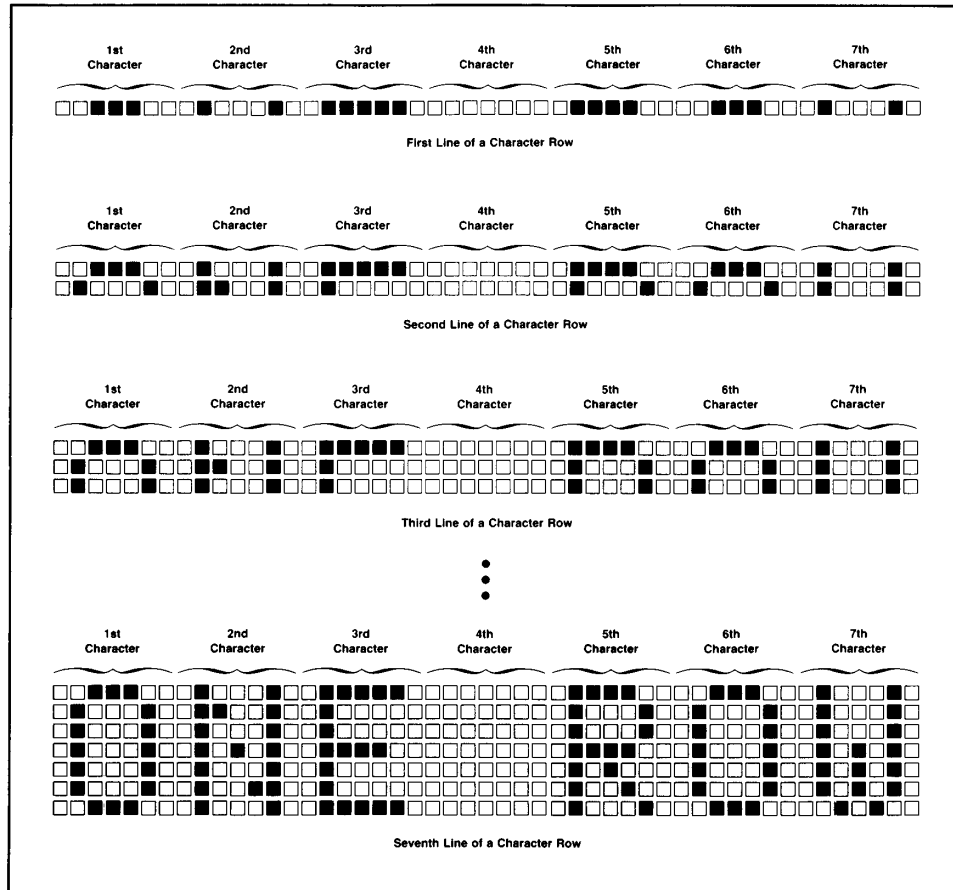


Figure 4.1.0 8276 Row Display

4.2 CRT Timing

The 8276 provides two timing outputs for controlling the CRT. The Horizontal Retrace Timing and Control (HRTC) and Vertical Retrace Timing and Control (VRTC) signals are used for synchronizing the CRT horizontal and vertical oscillators. A third output, VSP (Video Suppress), provides a signal to the dot timing logic to blank the video signal during the horizontal and vertical retraces. LTEN (Light Enable) is used to provide the ability to force the

video output high regardless of the state of the VSP signal. This feature is used to place the cursor on the screen and to control attribute functions.

RVV (Reverse Video) output, if enabled, will cause the system to invert its video output. The fifth timing signal output, HLG (highlight) allows the flexibility to increase the CRT beam intensity to a greater than normal level.

COMMAND	NO. OF PARAMETER BYTES	NOTES
RESET	4	Display format parameters required
START DISPLAY	0	DMA operation parameters included in command
STOP DISPLAY	0	—
RED LIGHT PEN	2	—
LOAD CURSOR	2	Cursor X, Y position parameters required
ENABLE INTERRUPT	0	—
DISABLE INTERRUPT	0	—
PRESET COUNTERS	0	Clears all internal counters

Figure 4.1.1 8276 Instruction Set

4.3 Special Functions

4.3.1 Special Codes

The 8276 recognizes four special codes that may be used to reduce memory, software, or system controller overhead. These characters are placed within the display memory by the system controller. The 8276 performs certain tasks when these codes are received in its row buffer memory.

- 1) *End of Row Code* — Activates VSP. VSP remains active until the end of the line is reached. While VSP is active the screen is blanked.
- 2) *End Of Row-Stop Buffer Loading Code* — Causes the Buffer Ready control logic to stop requesting buffer transfers for the rest of the row. It affects the display the same as End of Row Code.
- 3) *End Of Screen Code* — Activates VSP. VSP remains active until the end of the frame is reached.

4) *End Of Screen-Stop Buffer Loading Code* — Causes the Buffer Ready control logic to stop requesting buffer transfers until the end of the frame is reached. It affects the display the same way as the End of Screen code.

4.3.4 Programmable Buffer Loading Control

The 8276 can be programmed to request 1, 2, 4, or 8 characters per Buffer load. The interval between loads is also programmable. This allows the designer the flexibility to tailor the buffer transfer overhead to fit the system needs.

Each scan line requires 63.5 microseconds. A character line consists of 10 scan lines and takes 635 microseconds to form. The 8276 row buffer must be filled within the 635 microseconds or an under run condition will occur within the 8276 causing the screen to be blanked until the next vertical retrace. This blanking will be seen as a flicker in the display.

5.0 DESIGN BACKGROUND

A fully functional, microcontroller-based CRT terminal was designed and constructed using the 8051 and the 8276. The terminal has many of the functions that are found in commercially available low cost terminals. Sophisticated features such as programmable keys can be added easily with modest amounts of software.

The 8051's functions in this application note include: up to 9.6K BAUD full duplex serial transmission; decoding special messages sent from the host computer; scanning, debouncing, and decoding a full function keyboard; writing to the 8276 row buffer from the display RAM without the need for a DMA controller; and scrolling the display.

The 8276 CRT controller's functions include: presenting the data to the character generator; providing the timing signals needed for horizontal and vertical retrace; and providing blanking and video information.

5.1 Design Philosophy

Since the device count relates to costs, size, and reliability of a system, arriving at a minimum device count without degrading the performance was a driving force for this application note. LSI devices were used where possible to maintain a low chip count and to make the design cycle as short as possible.

PL/M-51 was chosen to generate the majority of the software for this application because it models the human thought process more closely than assembly language. Consequently it is easier and faster to write programs using PL/M-51 and the code is more likely to be correct because less chance exists to introduce errors.

PL/M-51 programs are easier to read and follow than assembly language programs, and thus are easier to modify and customize to the end user's application. PL/M-51 also offers lower development and maintenance costs than assembly language programming.

PL/M-51 does have a few drawbacks. It is not as efficient in code generation relative to assembly language and thus may also run slower.

This application note uses the 8051's interrupts to control the servicing of the various peripherals. The speed of the main program is less critical if interrupts are used. In the last two application notes on terminal controllers, a criterion of the system was the time required for receiving an incoming serial byte, decoding it, performing the function requested, scanning the keyboard, debouncing the keys, and transmitting the decoded ASCII code must be less than the vertical refresh time. Using the 8051 and its interrupts makes this time constraint irrelevant.

5.2 System Target Specifications

The design specifications for the CRT terminal design is as follows:

Display Format

- 80 characters/display row
- 25 display lines

Character Format

- 5 × 7 character contained within a 7 × 10 frame
- First and seventh columns blanked
- Ninth line cursor position
- Programmable delay blinking underline cursor

Control Characters Recognized

- Backspace
- Linefeed
- Carriage Return
- Form Feed

Escape Sequences Recognized

- ESC A, Cursor up
- ESC B, Cursor down
- ESC C, Cursor right
- ESC D, Cursor left
- ESC E, Clear screen
- ESC F, Move addressable cursor
- ESC H, Home cursor
- ESC J, Erase from cursor to the end the screen
- ESC K, Erase the current line

Characters Displayed

- 96 ASCII Alphanumeric Characters

Characters Transmitted

- 96 ASCII Alphanumeric Characters
- ASCII Control Character Set
- ASCII Escape Sequence Set
- Auto Repeat

Display Memory

- 2K × 8 static RAM

Data Rate

- Variable rate from 150 to 9600 BAUD

CRT Monitor

- Ball Bros TV-12, 12MHZ Black and White

Keyboard

- Any standard undecoded keyboard (2 key lock-out)
- Any standard decoded keyboard with output enable pin
- Any standard decoded serial keyboard up to 150 BAUD

Scrolling Capability

Compatible With Wordstar

6.0 SYSTEM DESCRIPTION

A block diagram of the CRT terminal is shown in figure 6.0.0. The diagram shows only the essential system features. A detailed schematic of the CRT terminal is contained in the Appendix 7.1.

The "brains" of the CRT terminal is the 8051 microcontroller. The 8276 is the CRT controller in the system, and a 2716 EPROM is used as the character generator. To handle the high speed portion of the CRT, the 8276 is surrounded by a handful of TTL devices. A 2K × 8 static RAM was used as the display memory.

Following the system reset, the 8276 is initialized for cursor type, number of characters per line, number of lines, and character size. The display RAM is initialized to all "spaces" (ASCII 20H). The 8051 then writes the "start display" command to the 8276. The local/line input is sampled to determine the terminal mode. If the terminal is on-line, the BAUD rate switches are read and the serial port is set up for full duplex UART mode. The processor then is put into a loop waiting to service the serial port fifo or the 8276.

The serial port is programmed to have the highest priority interrupt. If the serial port generates an interrupt, the processor reads the buffer, puts the character in a generated fifo that resides in the 8051's internal RAM, increments the fifo pointer, sets the serial interrupt flag and returns.

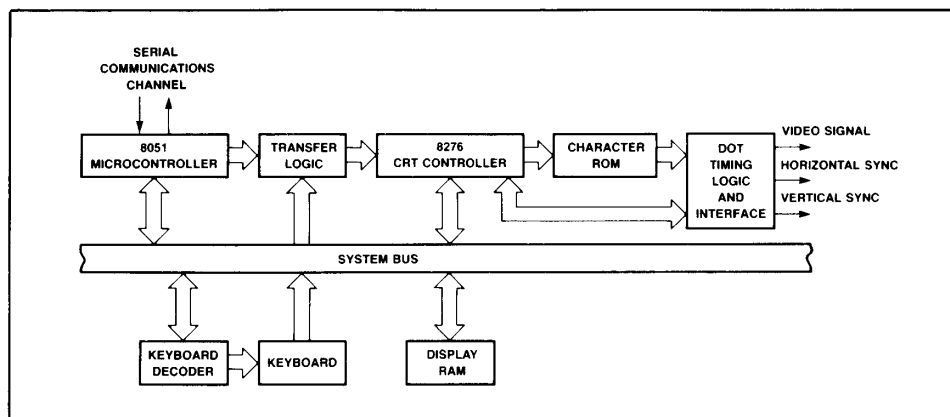


Figure 6.0.0 CRT Terminal Controller Block Diagram

The main program determines if it is a displayable character, a Control word or an ESC sequence and either puts the character in the display buffer or executes the appropriate command sent from the host computer.

If the 8276 needs servicing, the 8051 fills the row buffer for the CRT display's next line. If the 8276 generates a vertical retrace interrupt, the buffer pointers are reloaded with the display memory location that corresponds to the first character of the first display line on the CRT. The vertical retrace also signals the processor to read the keyboard for a key closure.

Three general cases can be explored; reading and writing the display RAM, writing to the 8276 row buffers, and reading and writing the 8276's control registers.

As mentioned previously the 8051 fills the 8276 row buffer without the need of a DMA controller. This is accomplished by using a Quad 2-input multiplexor (Figure 6.1.0) as the transfer logic shown in the block diagram. The address line, P2.3, is used to select either of the two inputs. When the address line is low the RD and WR lines perform their normal functions, that is read and write the

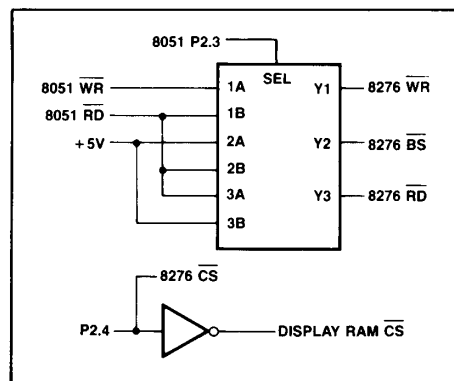
6.1 Hardware Description

The following section describes the unique characteristics of this design.

6.1.1 Peripheral Address Map

The display RAM, 8276 registers, and the 8276 row buffers are memory mapped into the external data RAM address area. The addresses are as follows:

Read and Write External Display RAM —	Address 1000H to 17CFH
Write to 8276 row buffers from Display RAM —	Address 1800H to 1FCFH
Write to 8276 Command Register (CREG) —	Address 0001H
Write to 8276 Parameter Register (PREG) —	Address 0000H
Read from 8276 Status Register (SREG) —	Address 0001H

Figure 6.1.0
Simplified Version Of The Transfer Logic

8276 or the external display RAM depending on the states of their respective chip selects. If the address line is high, the 8051 RD line is transformed into BS and WR signals for the 8276. While holding the address line high, the 8051 executes an external data move (MOVX) from the display RAM to the accumulator which causes the display RAM to output the addressed byte onto the data bus. Since the multiplexor turns the same 8051 RD pulses into BS and WR pulses to the 8276, the data bus is thus read into the 8276 as a Buffer transfer. This scheme allows 80 characters to be transferred from the display RAM into the 8276 within the required character line time of 635 microseconds. The 8051 easily meets this requirement by accomplishing the task within 350 microseconds.

6.1.2 Scanning The Keyboard

Throughout this project, provision have been made to make the overall system flexible. The software has been written for various keyboards and the user simply needs to link different program modules together to suit their needs.

6.1.2.1 Undecoded Keyboard

Incorporating an undecoded keyboard controller into the other functions of the 8051 shows the flexibility and over all CPU power that is available. The keyboard in this case is a full function, non-buffered 8×8 matrix of switches for a total of 64 possible keys. The 8 send lines are connected to a 3-to-8 open-collector decoder as shown in Figure 6.1.1. Three high order address lines from the 8051 are the decoder inputs. The enabling of the decoder is accomplished through the use of the PSEN signal from the 8051 which makes the architecture of the separate address space for the program memory and the external data RAM work for us to eliminate the need to decode addresses externally. The move code (MOVX) instruction allows each scan line of the keyboard to be read with one instruction.

The keyboard is read by bringing one of the eight scan lines low sequentially while reading the return lines which are pulled high by an external resistor. If a switch is

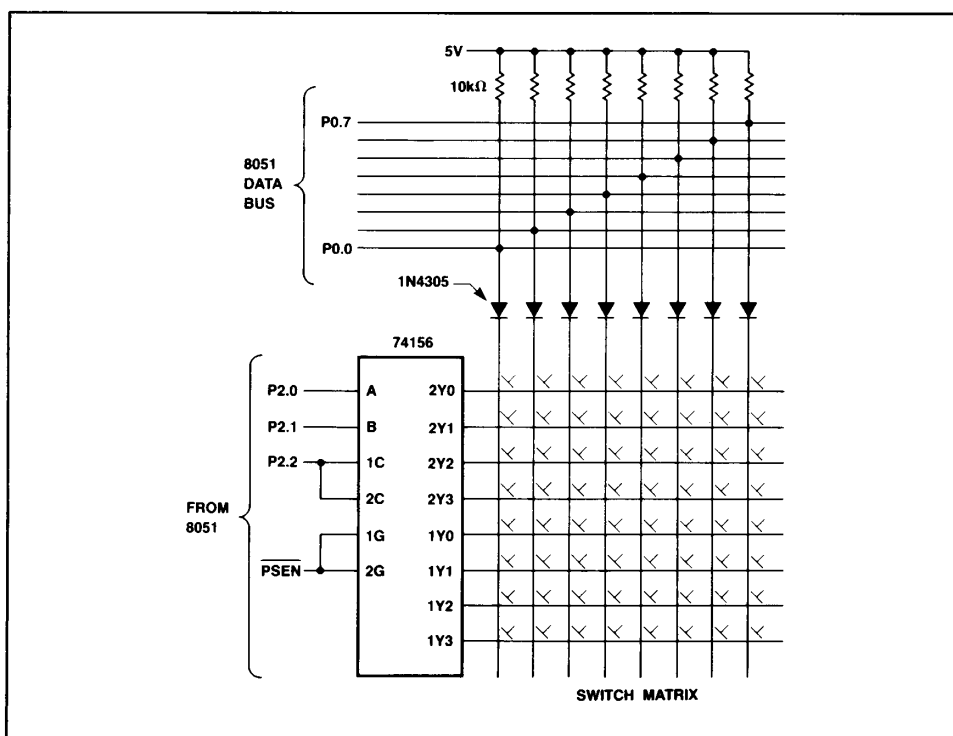


Figure 6.1.1 Keyboard

closed, the data bus line is connected through the switch to the low output of the decoder and one of the data bus lines will be read as a 0. By knowing which scan line detected a key closure and which data bus line was low, the ASCII code for that key can easily be looked up in a matrix of constants. PL/M-51 has the ability to handle arrays and structured arrays, which makes the decoding of the keyboard a trivial task.

Since the Shift, Cap Lock, and Control keys may change the ASCII code for a particular key closure, it is essential to know the status of these pins while decoding the keyboard. The Shift, Cap Lock, and Control keys are therefore not scanned but are connected to the 8051 port pins where they can be tested for closure directly.

The 8 receive lines are connected to the data bus through germanium diodes which chosen for their low forward voltage drop. The diodes keep the keyboard from interfering with the data bus during the times the keyboard is not being read. The circuit consisting of the 3-to-8 decoder and the diodes also offers some protection to the 8051 from possible Electrostatic Discharge (ESD) damage that could be transmitted through the keyboard.

6.1.2.2 Decoded Keyboard

A decoded keyboard can easily be connected to the system as shown in Figure 6.1.2. Reading the keyboard can be evoked either by interrupts or by software polling.

The software to periodically read a decoded keyboard was not written for this application note but can be accomplished with one or two PL/M-51 statements in the `READER` routine.

A much more interesting approach would be to have the servicing of the keyboard be interrupt driven. An additional external interrupt is created by configuring timer/counter 0 into an event counter. The event counter is

initialized with the maximum count. The keyboard controller would inform the 8051 that a valid key has been depressed by pulling the input pin `T0` low. This would overflow the event counter, thus causing an interrupt. The interrupt routine would simply use a `MOVC` (`PSEN` is connected to the output enable pin of the keyboard controller) to read the contents of the keyboard controller onto the data bus, reinitialize the counter to the maximum count and return from the interrupt.

6.1.2.3 Serial Decoded Keyboard

The use of detachable keyboards has become popular among the manufacturers of keyboards and personal computers. This terminal has provisions to use such a keyboard.

The keyboard controller would scan the keyboard, debounce the key and send back the ASCII code for that key closure. The message would be in an asynchronous serial format.

The flowchart for a software serial port is shown in Figure 6.1.3. An additional external interrupt is created as discussed for the decoded keyboard but the use in this case would be to detect a start bit. Once the beginning of the start bit has been detected, the timer/counter 0 is configured to become a timer. The timer is initialized to cause an interrupt one-half bit time after the beginning of the start bit. This is to validate the start bit. Once the start bit is validated, the timer is initialized with a value to cause an interrupt one bit time later to read the first data bit. This process of interrupting to read a data bit is repeated until all eight data bits have been received. After all 8 data bits are read, the software serial port is read once more to detect if a stop bit is present. If the stop bit is not present, an error flag is set, all pointers and flags are reset to their initial values, and the timer/counter is reconfigured to an event counter to detect the next start bit. If the stop bit is present, a valid flag is set and the flags and counter are reset as previously discussed.

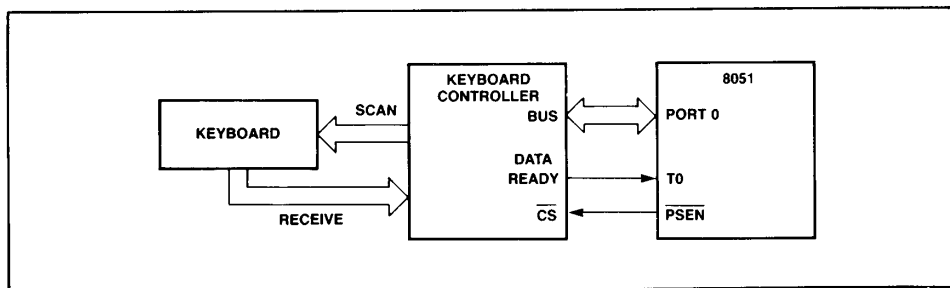


Figure 6.1.2 Using A Decoded Keyboard

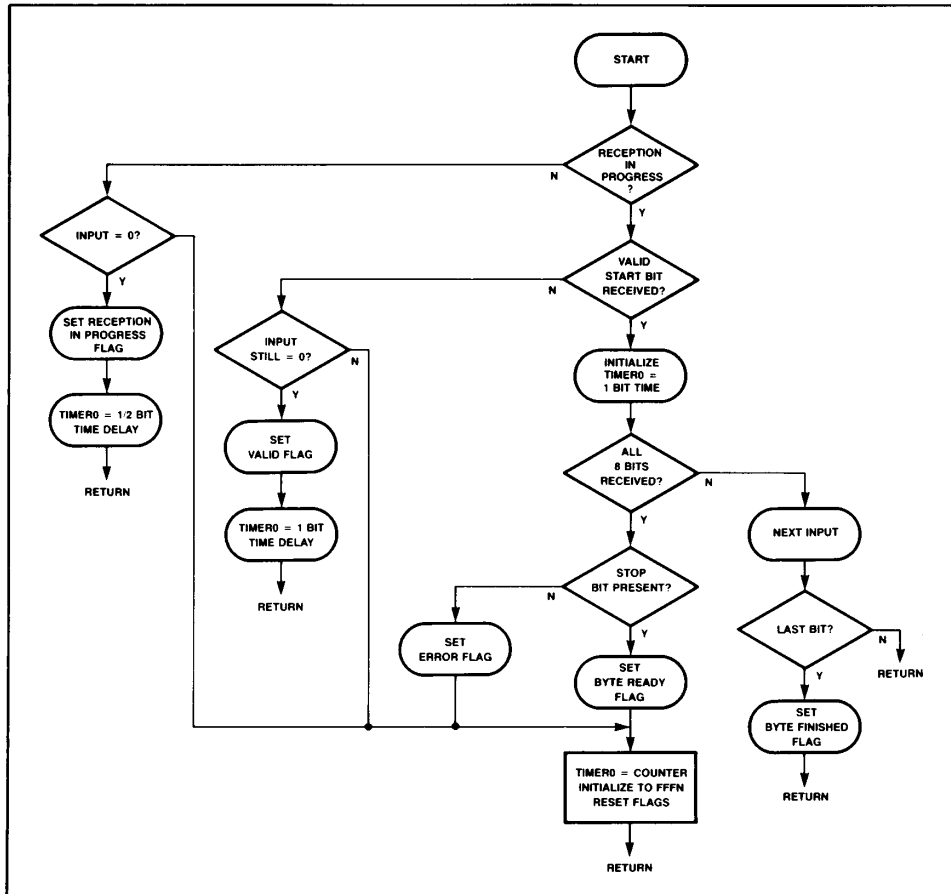


Figure 6.1.3 Flowchart for the Software Serial Port

6.1.4 System Timings

The requirements for the BALL BROTHERS, TV-12 monitor's operation is shown in table 6.1.0. From the monitor's parameters, the 8276 specifications and the system target specifications the system timing is easily calculated.

The 8276 allows the vertical retrace to be only an integer multiple of the horizontal character lines. Twenty-five display lines and a character frame of 7×10 are required from the target specification which will require 250 horizontal lines. If the horizontal frequency is to be within

500 Hz of 15,750 Hz, we must choose either one or two character line times for horizontal retrace. To allow for a little more margin at the top and bottom of the screen, two character line times was chosen for the vertical retrace. This choice yields $250 + 20 = 270$ total character lines per frame. Assuming 60 Hz vertical retrace frequency:

$$60 \text{ Hz} \times 270 = 16,200 \text{ Hz horizontal frequency}$$

and

$$1/16,200 \text{ Hz} \times 20 \text{ horizontal sync times} = 1.2345 \text{ milliseconds}$$

Table 6.1.0 CRT Monitor's Operational Requirements

PARAMETER	RANGE
Vertical Blanking Time (VRTC)	800 μ sec nominal
Vertical Drive Pulsewidth	300 μ sec \leq PW \leq 1.4 ms
Horizontal Blanking Time (HRTC)	11 μ sec nominal
Horizontal Drive Pulsewidth	25 μ sec \leq PW \leq 30 μ sec
Horizontal Repetition Rate	15,750 \pm 500 pps

The 1.2345 milliseconds of retrace time meets the nominal VRTC and vertical drive pulse width time of .3mSec to 1.4mSec for the Ball monitor.

The next parameter to find is the horizontal retrace time which is wholly dependent on the monitor used. Usually it lies between 15 and 30 percent of the total horizontal line time.

Since most designs display a fixed number of characters per line it is useful to express the horizontal retrace time as a given number of character times. In this design, 80 characters are displayed, and it was experimentally found that 20 character times for the horizontal retrace gave the best results. It should be noted if too much time was given for retrace, there would be less time to display the characters and the display would not fill out the screen. Conversely, if not enough time is given for retrace, the characters would seem to run off the screen.

One hundred character times per complete horizontal line means that each character needs:

$$(1/16,200 \text{ Hz}) / 100 \text{ character times} = 617.3 \text{ nanoseconds}$$

If we multiply the 20 character times needed to retrace by 617.3 nanoseconds needed for each character, we find 12.345 microseconds are allocated for retrace. This value falls short of the 25 to 30 microseconds required by the horizontal drive of the Ball monitor. To correct for this, a 74LS123 one-shot was used to extend the horizontal drive pulse width.

The dot clock frequency is easy to calculate now that we know the horizontal frequency. Since each character is formed by seven dots in the horizontal axis, the dot clock period would be the character clock (617.3 nanoseconds) divided by the 7 which is equal to 11.34 MHz. The basic dot timing and CRT timing are shown in the Appendix.

6.2 Software Description

6.2.1 Software Overview

The software for this application was written in a "foreground-background" format. The background programs are all interrupt driven and are written in assembly language due to time constraints. The foreground programs are for the most part written in PL/M-51 to ease the programming effort. A number of subroutines are written in assembly language due to time constraints during execution. Subroutines such as clearing display lines, clearing the screen, and scanning the keyboard require a great deal of 16 bit adds and compares and would execute much slower and would require more code space if written in PL/M-51. The background and foreground programs talk to each other through a set of flags. For example, the PL/M-51 foreground program tests "SERIAL\$INT" to determine if a serial port interrupt had occurred and a character is waiting to be processed.

6.2.2 The Background Program

Two interrupt driven routines, VERT and BUFFER, (see Fig. 6.2.0) request service every 16.67 milliseconds and 617 microseconds respectively. VERT's request comes during the last character row of the display screen. This routine resets the buffer pointers to the first CRT display line in the display memory. VERT is also used as a time base for the foreground program. VERT sets the flag, SCAN, to tell the foreground program (PL/M-51) that it is time to scan the Keyboard. VERT also increments a counter used for the delay between transmitting characters in the AUTO\$REPEAT routine.

The BUFFER routine is executed once per character row. BUFFER uses the multiplexor discussed earlier to fill the 8276's row buffer by executing 80 external data moves and incrementing the Data Pointer between each move.

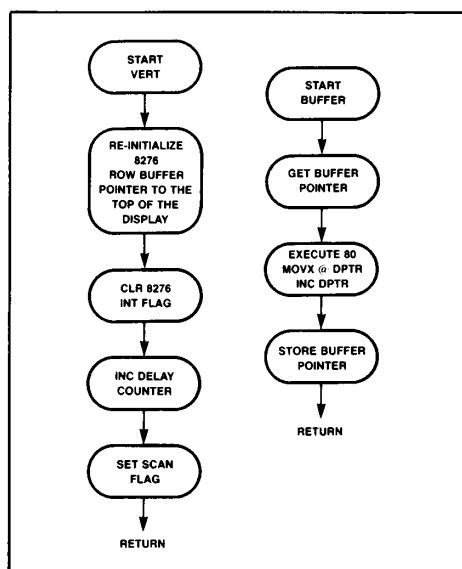


Figure 6.2.0 Flowcharts For VERT and BUFFER Routine

The MOVX reads the display RAM and writes the character into the row buffer during the same instruction.

SERBUF is an interrupt driven routine that is executed each time a character is received or transmitted through the on-chip serial port. The routine first checks if the interrupt was caused by the transmit side of the serial port, signaling that the transmitter is ready to accept another character. If the transmitter caused the interrupt, the flag "TRANSMIT\$INT" is set which is checked by the foreground program before putting a character in the buffer for transmission.

If the receiver caused the interrupt, the input buffer on the serial port is read and fed into the fifo that has been manufactured in the internal RAM and increments the fifo pointer "FIFO." The flag "SERIAL\$INT" is then set, telling the foreground program that there is a character in the fifo to be processed. If the read character is an ESC character, the flag "ESCSEQ" is set to tell the foreground program that an escape sequence is in the process of being received.

6.2.3 The Foreground Program

The foreground program is documented in the Appendix. The foreground program starts off by initializing the 8276

as discussed earlier. After all variables and flags are initialized, the processor is put into a loop waiting for either VERT to set SCAN so the program can scan the keyboard, or for the serial port to set SERIAL\$INT so the program can process the incoming character.

The vertical retrace is used to time the delay between keyboard scans. When VERT gets set, the assembly language routine READER is called. READER scans the keyboard, writing each scan into RAM to be processed later. READER controls two flags, KEY0 and SAME. KEY0 is set when all 8 scans determine that no key is pressed. SAME is set when the same key that was pressed last time the keyboard was read is still pressed.

After READER returns execution to the main program, the flags are tested. If the KEY0 flag is set the main program goes back to the loop waiting for the vertical retrace or a serial port interrupt to occur. If the SAME flag is set the main program knows that the closed key has been debounced and decoded so it sends the already known ASCII code to the AUTO\$REPEAT routine which determines if that character should be transmitted or not.

If KEY0 and SAME are not set, signifying that a key is pressed but it is not the same key as before, the foreground program determines if the results from the scan are valid. First all eight scans are checked to see if only one key was closed. If only one key is closed, the ASCII code is determined, modified if necessary by the Shift, Cap Lock, or Control keys. The NEWSKEY and VALID flags are then set. The next time READER is called, if the same key is still pressed, the SAME flag will be set, causing the AUTO\$REPEAT subroutine to be called as just discussed. Since the keyboard is read during the vertical retrace, 16.67 milliseconds has elapsed between the detection of the pressed key and reverifying that the key is still pressed before transmitting it, thus effectively debouncing the key.

The AUTO\$REPEAT routine is written to transmit any key that the NEWSKEY flag is set for. The counter that is incremented each time the vertical refresh interrupt is serviced causes a programmable delay between the first transmission and subsequent auto repeat transmission. Once the NEWSKEY character is sent, the counter is initialized. Each time the AUTO\$REPEAT routine is called, the counter is checked. Only when the counter overflows will the next character be transmitted. After the initial delay, a character will be transmitted every other time the routine is called as long as the key remains pressed.

6.2.3.1 Handling Incoming Serial Data

One of the criteria for this application note was to make the software less time dependent. By creating a fifo to store incoming characters until the 8051 has time to pro-

cess them, software timing becomes less critical. This application note uses up to 8 levels of the fifo at 9.2KBAUD, and 1 level at 4.8KBAUD and lower. As discussed earlier, the interrupt service routine for the serial port uses the fifo to store incoming data, increments the fifo pointer, "FIFO", and sets SERIAL\$INT to tell the main program that the fifo needs servicing. Once the main program detects that SERIAL\$INT is set the routine DECIPHER is executed.

DECIPHER has three separate blocks; a block for decoding displayable characters, a block for processing Escape sequences, and a block for processing Control codes. Each block works on the fifo independently. Before exiting a block, the contents of the fifo are shifted up by the amount of characters that were processed in that particular block. The shifting of the characters insures that the beginning of the fifo contains the next character to be processed. FIFO is then decremented by the number of characters processed.

Let's look at this process more closely. Figure 6.2.1-A shows a representation of a fifo containing 5 characters. The first three characters in the fifo contain displayable characters, A, B, and C respectively with the last two characters being an ESC sequence for moving the cursor up one line (ESC A) and FIFO points to the next available location to be filled by the serial port interrupt routine, in this case, 5.

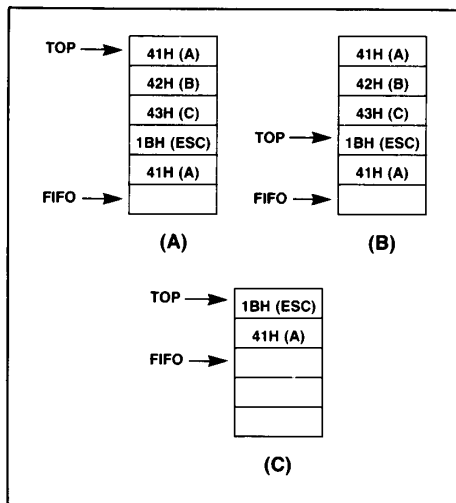


FIGURE 6.2.1 FIFO

When DECIPHER is executed, the first block begins looking at the first character of the fifo for a displayable character. If the character is displayable, it is placed into the display RAM and the software pointer "TOP" that points to the character that is being processed is incremented to the next character. The character is then looked at to see if it too is displayable and if it is, it's placed in the display RAM. The process of checking for displayable characters is continued until either the fifo is empty or a non-displayable character is detected. In our example, three characters are placed into the display RAM before a non-displayable character is detected. At this point the fifo looks like figure 6.2.1-B.

Before entering the next block, the remaining contents of the fifo between TOP, that is now pointing to 1BH and (FIFO-1) are moved up in the fifo by the amount of characters processed, in this example three. TOP is reset to 0 and FIFO is decremented by 3. The serial port interrupt is inhibited during the time the contents of the fifo and the pointers are being manipulated. The fifo now looks like figure 6.2.1-C.

The execution is now passed to the next block that processes ESC sequences. The first location of the fifo is examined to see if it is an ESC character (1BH). If not, the execution is passed to the next block of DECIPHER that processes Control codes. In this case the fifo does contain an ESC code. The flag ESC\$SEQ is checked to see if the 8051 is in the process of receiving an ESC sequence thus signifying that the next byte of the sequence has not been received yet. If the ESC\$SEQ is not set, the next character in the fifo is checked for a valid escape code and the proper subroutine is then called. The fifo contents are then shifted as discussed for the previous block. Due to the length of time that is needed to execute an ESC code sequence or a Control code, only one ESC code and/or Control code can be processed each time DECIPHER is executed.

If at the end of the DECIPHER routine, FIFO contains a 0, the flag SER\$INT is reset. If SER\$INT remains set, DECIPHER will be executed immediately after returning to the main program if SCAN had not been set during the execution of the DECIPHER routine, otherwise DECIPHER will be called after the keyboard is read.

6.2.4 Memory Pointers and Scrolling

The cursor always points to the next location in display memory to be filled. Each time a character is placed in the display memory, the cursor position needs to be tested to determine if the cursor should be incremented to the beginning of the next line of the display or simply moved to the next position on the current display line. The cursor position pointers are then updated in both the 8276 and the internal registers in the 8051.

When the 2000th character is entered into the display memory, a full display page has been reached signaling the need for the display to scroll. The memory pointer that points to the display memory that contains the first character of the first display line, LINE0, prior to scrolling contains 1800H which is the starting address of the display memory. Each scrolling operation adds 80 (50H) to LINE0 which will now point to the following row in memory as shown in figure 6.2.2-B. LINE0 is used during the vertical

refresh routine to re-initialize the pointers associated with filling the 8276 row buffers.

The display memory locations that were the first line of the CRT display now becomes the last line of the CRT display. Incoming characters are now entered into the display memory starting with 1800H, which is now the first character of the last line of the display screen.

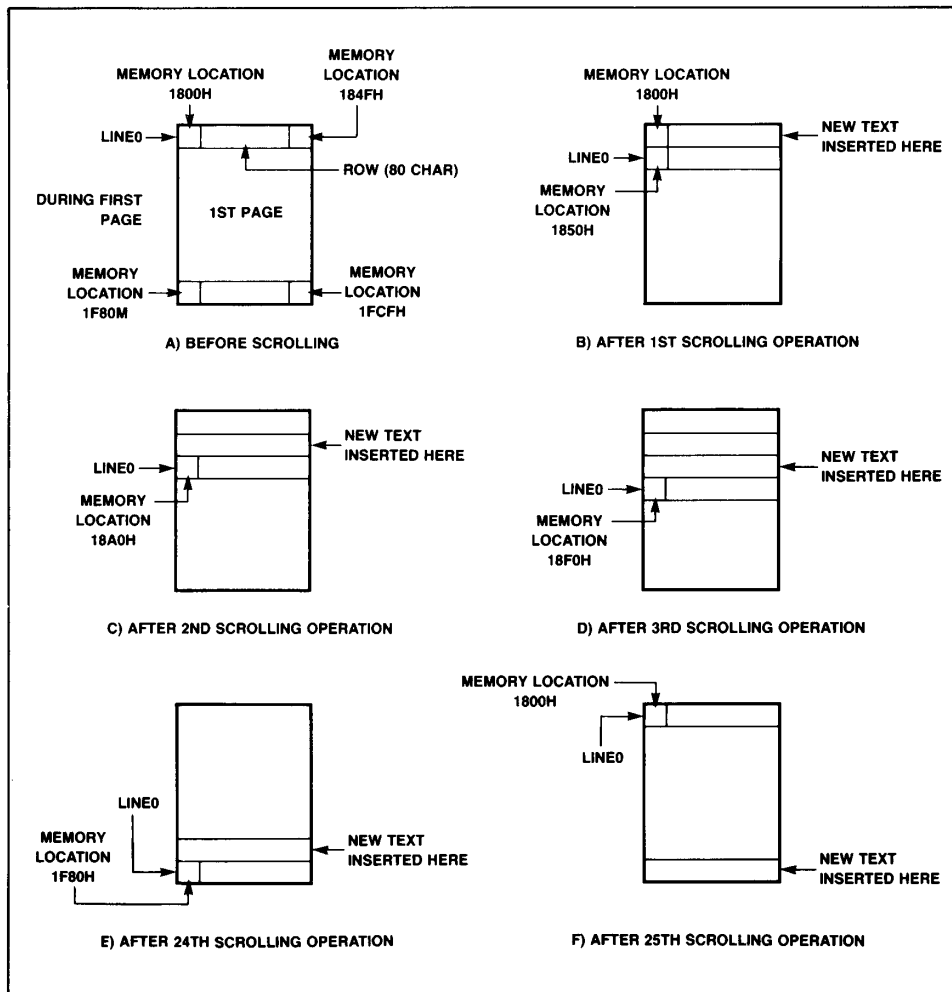


Figure 6.2.2 Pointer Manipulation During Scrolling

6.2.5 Software Timing

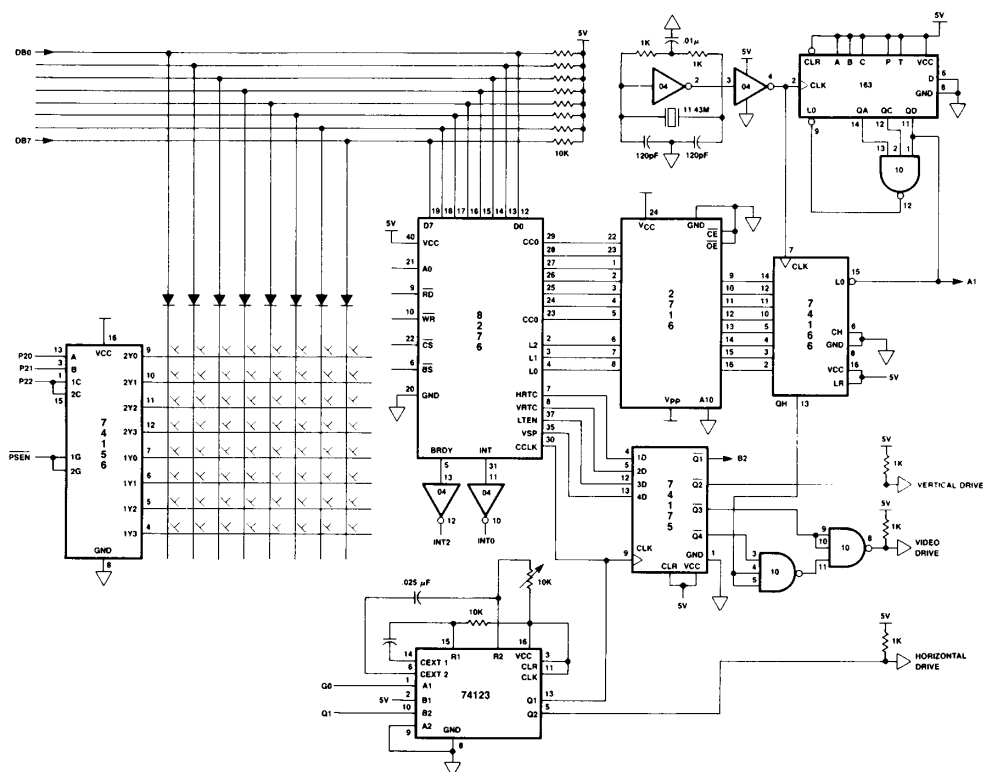
The use of interrupts to tie the operation of the foreground program to the real-time events of the background program has made the software timing non-critical for this system.

6.3 System Operation

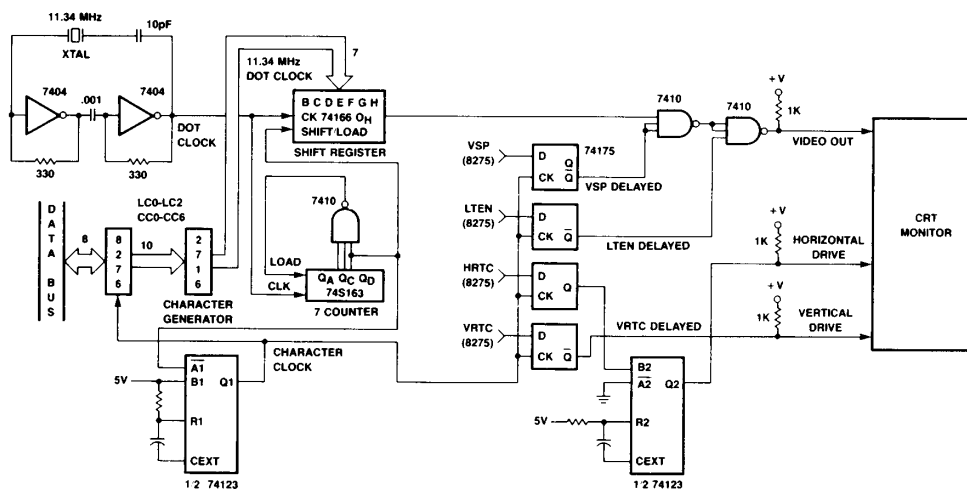
Following the system reset, the 8051 initializes all on-chip peripherals along with the 8276 and display ram. After initialization, the processor waits until the fifo has a character to process or is flagged that it is time to scan the keyboard. This foreground program is interrupted once every 617 microseconds to service the 8276 row buffers. The 8051 is also interrupted each 16.67 milliseconds to re-initialize LINE0 and to flag the foreground program to read the keyboard.

As discussed earlier, a special technique of rapidly moving the contents of the display RAM to the 8276 row buffers without the need of a DMA device was employed. The characters are then synchronously transferred to the character generator via CC0-CC6 and LC0-LC2 which are used to display one line at a time. Following the transfer of the first line to the dot timing logic, the line count is incremented and the second line is selected. This process continues until the last line of the character is transferred.

The dot timing logic latches the output of the character ROM in a parallel in, serial out synchronous shift register. The shift register's output constitutes the video information to the CRT.

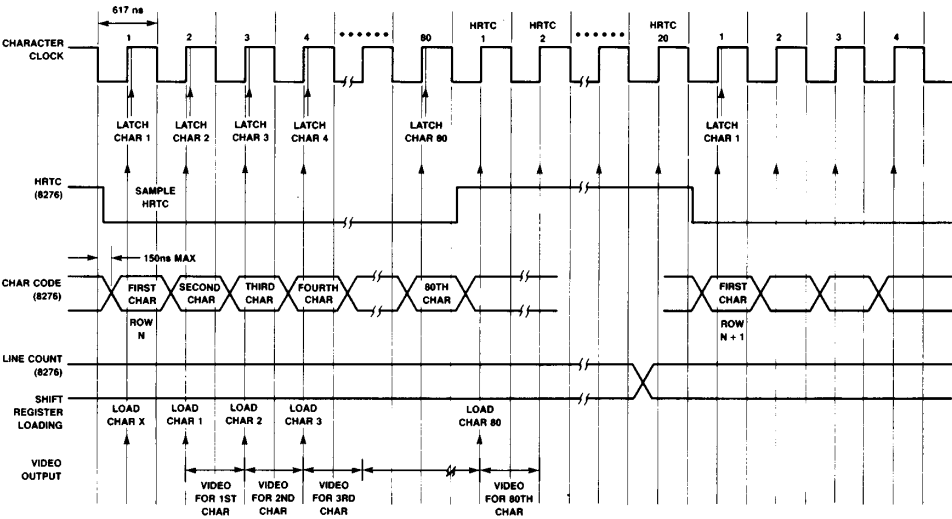


Appendix 7.2 Dot Timing





Appendix 7.3 CRT System Timing



Appendix 7.4 Escape/Control/Display Character Summary

BIT	CONTROL CHARACTERS				DISPLAYABLE CHARACTER				ESCAPE SEQUENCE					
	000	001	010	011	100	101	110	111	010	011	100	101	110	111
0000	NUL	DLE P	SP			P		P						
0001	SOH A	DC1 Q	!	:	A	Q	A	Q				A		
0010	STX B	DC2 R	-	2	B	R	B	R				B		
0011	ETX C	DC3 S	=	3	C	S	C	S				C		
0100	EOT D	DC4 T	\$	4	D	T	D	T				D		
0101	ENQ E	NAK	%	5	E	U	E	U			CLR	E		
0110	ACK F	SYN V	&	6	F	V	F	V						
0111	BEL G	ETB W	'	7	G	W	G	W						
1000	SS H	CAN X	(8	H	X	H	X			HOME	H		
1001	HT I	EM Y)	9	I	Y	I	Y						
1010	LF J	SUB Z	*	:	J	Z	J	Z			EOS	I		
1011	VT K	ESC	+	:	K	[K				EL	J		
1100	FF L	FS	,		L		L							
1101	CR M	GS	-	=	M]	M							
1110	SO N	RS	.		N	^	N							
1111	S1	US -	/	?	O	_	O							

NOTE: Shaded blocks — functions terminal will react to. Others can be generated but are ignored upon receipt.



Appendix 7.5 Character Generator

As previously mentioned, the character generator used in this terminal is a 2716 EPROM. A 1K by 8 device would have been sufficient since a 128 character 5 by 7 dot matrix only requires 8K of memory. A custom character set could have been stored in the second 1K bytes of the 2716. Any of the free I/O pins on the 8051 could have been used to switch between the character sets.

The three low-order line count outputs (LC0-LC2) from the 8276 are connected to the three low-order address lines of the character generator. The CC0-CC6 output lines are connected to the A3-A9 lines of the character generator.

The output of the character generator is loaded into the shift register. The serial output of the shift register is the video output to the CRT.

Let's assume that the letter "E" is to be displayed. The ASCII code for "E" (45H) is presented to the address lines A2-A9 of the character generator. The scan lines (LC0-LC2) will now count from 0 to seven to form the character as shown in Figure 7.5.0. The same procedure is used to form all 128 possible characters. For reference Appendix 7.6 contains the HEX dump of the character generator used in this terminal.

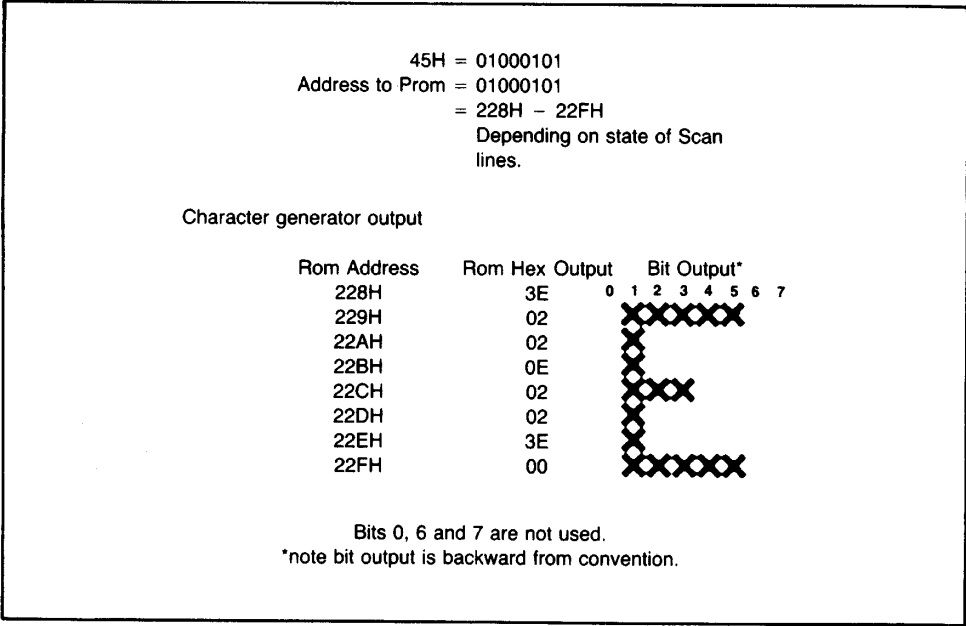


Figure 7.5.0 Character Generator



25

Appendix 7.7 Composite Video

In this design it was assumed that the CRT monitor required a separate horizontal drive, vertical drive, and video input. Many monitors require a composite video signal. The schematic shown in Figure 7.7.0 illustrate how to generate a composite video from the output of the 8276.

The dual one-shots are used to provide a small delay and the proper horizontal and vertical pulse to the composite video monitor. The delay introduced in the horizontal and vertical timing is used to center the display. The 7486 is used to mix the vertical and horizontal retrace. Q1 mix the video and retrace signals along with providing the proper D.C. levels.

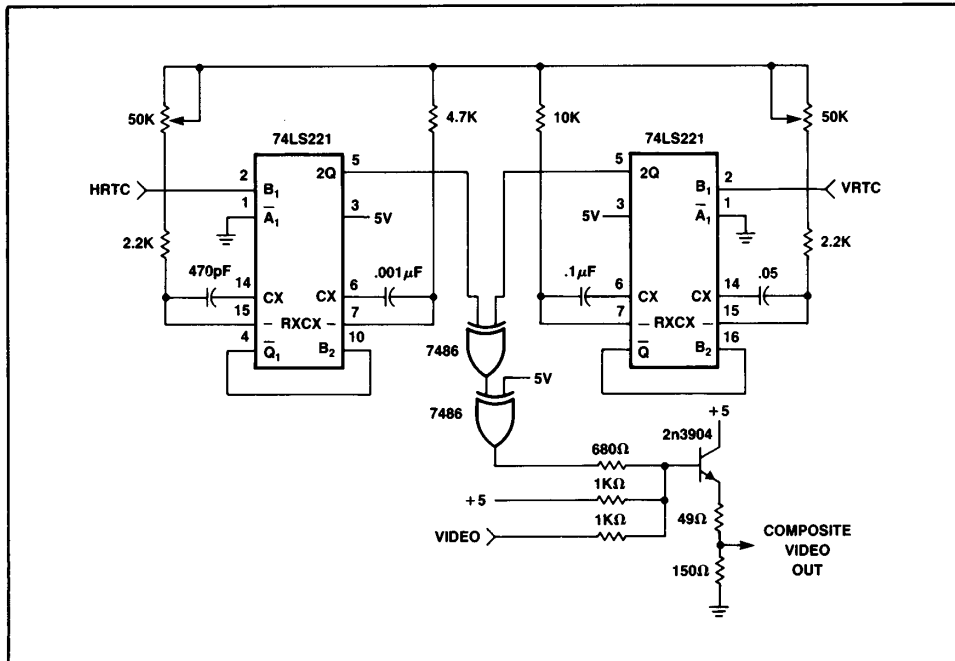


Figure 7.7.0 Composite Video

Appendix 7.8 Software Documentation

```

/*****
*****
***** SOFTWARE DOCUMENTATION FOR THE 8051 *****
***** TERMINAL CONTROLLER APPLICATION NOTE *****
*****
*****
*****/

MEMORY MAP ASSOCIATED WITH PERIPHERAL DEVICES (USING MOVX):

8051 WR AND READ DISPLAY RAM- ADDRESS 1000H TO 17CFH
8051 WR DISPLAY RAM TO THE 8276- ADDRESS 1800H TO 1FCFH
8276 COMMAND ADDRESS- ADDRESS 0001H
8276 PARAMETER ADDRESS- ADDRESS 0000H
8276 STATUS REGISTER- ADDRESS 0001H

MEMORY MAP FOR READING THE KEYBOARD (USING MOVX):

KEYBOARD ADDRESS- ADDRESS 10FFH TO 17FFH

/***** START MAIN PROGRAM *****/

/* BEGIN BY PUTTING THE ASCII CODE FOR BLANK IN THE DISPLAY RAM*/

INIT:
{FILL 2000 LOCATIONS IN THE DISPLAY RAM WITH SPACES (ASCII 20H)}

/* INITIALIZE POINTERS, RAM BITS, ETC. */

{INITIALIZE POINTERS AND FLAGS}
{INITIALIZE TOP OF THE CRT DISPLAY "LINE0"=1800H}
{INITIALIZE 8276 BUFFER POINTER "RASTER" =1800H}
{INITIALIZE DISPLAY$RAM$POINTER=0000H}

/* INITIALIZE THE 8276 */

{ RESET THE 8276}
{INITIALIZE 8276 TO 80 CHARACTER/ROW }
{INITIALIZE 8276 TO 25 ROWS PER FRAME}
{INITIALIZE 8276 TO 10 LINES PER ROW}
{INITIALIZE 8276 TO NON-BLINKING UNDERLINE CURSER}
{INITIALIZE CURSER TO HOME POSITION (00,00) (UPPER LEFT HAND CORNER)}
{START DISPLAY}
{ENABLE 8276 INTERRUPT}

/* SET UP 8051 INTERRUPTS AND PRIORITIES */

{SERIAL PORT HAS HIGHEST INTERRUPT PRIORITY}
{EXTERNAL INTERRUPTS ARE EDGE SENSITIVE}
{ENABLE EXTERNAL INTERRUPTS}

```

```

/*PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A
SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT
WILL BE TRANSMITTED TO THE HOST COMPUTER.*/

SCANNER:
{ENABLE 8051 GLOBAL INTERRUPT BIT}

/* PROGRAMMABLE DELAY FOR THE CURSER BLINK */
IF {30 VERTICAL RETRACE INTERRUPTS HAVE OCCURRED (CURSER$COUNT=1FH)} THEN
DO;
  {COMPLEMENT CURSER$ON}
  {CLEAR CURSER$COUNT}
  IF {CURSER IS TO BE OFF (CURSER$ON=0)} THEN {MOVE CURSER OFF THE SCREEN}
  CALL LOAD$CURSER;
END;

IF {THE LOCAL$LINE SWITCH HAS CHANGED STATE} THEN
DO;
  IF {IN LOCAL MODE} THEN {DISABLE SERIAL PORT INTERRUPT}
  ELSE CALL CHECK$BAUD$RATE;
END;

DO WHILE {INBETWEEN VERTICAL REFRESHES}
  IF {THE FIFO HAS A CHARACTER TO PROCESS (SERIAL$INT=1)} THEN CALL DECIPHER;
END;

CALL READER;

IF {THE PRESENT PRESSED KEY IS EQUAL TO THE LAST KEY PRESSED AND VALID=1} THEN
  CALL AUTO$REPEAT;
ELSE
DO;
  IF {A KEY IS PRESSED BUT NOT THE SAME ONE AS THE LAST KEYBOARD SCAN} THEN
  DO;
    IF {ONLY ONE KEY IS PRESSED} THEN
      {GET THE ASCII CODE FOR IT}
      {SET NEWSKEY AND VALID FLAGS}
    ELSE {RESET VALID AND NEWSKEY FLAGS}
  END;
  ELSE {THE KEYBOARD MUST NOT HAVE A KEY PRESSED SO RESET VALID$KEY AND NEWSKEY FLAGS}
END;

GOTO SCANNER;
END;

/* PROCEDURE AUTO$REPEAT: THIS PROCEDURE WILL PERFORM AN AUTO REPEAT FUNCTION
BY TRANSMITTING A CHARACTER EVERY OTHER TIME THIS ROUTINE IS CALLED.
THE AUTO REPEAT FUNCTION IS ACTIVATED AFTER A FIXED DELAY PERIOD AFTER THE
FIRST CHARACTER IS SENT*/

AUTO$REPEAT:
IF {THE KEY PRESSED IS NEW (NEWSKEY=1)} THEN
DO;
  {CLEAR THE DIVIDE BY TWO COUNTER "TRANSMIT$TOGGLE"}
  {INITIALIZE THE DELAY COUNTER "TRANSMIT$COUNT" TO 000H}
  CALL TRANSMIT;
  {CLEAR NEWSKEY}
END;
/* FIRST CHARACTER */

```

```

ELSE
DO;
  IF {TRANSMIT$COUNT IS NOT EQUAL TO 0} THEN
  DO;
    {INCREMENT TRANSMIT$COUNT}
    IF TRANSMIT$COUNT=OFFH THEN /*DELAY BETWEEN FIRST CHARACTER AND THE SECOND */
    DO;
      CALL TRANSMIT; /*SECOND CHARACTER */
      {CLEAR TRANSMIT$COUNT}
    END;
  END;
ELSE
DO;
  {TURN THE CURSER ON DURING THE AUTO REPEAT FUNCTION}
  IF TRANSMIT$TOGGLE = 1 THEN /* 2 VERT FRAMES BETWEEN 3RD TO NTH CHARACTER */
  CALL TRANSMIT; /* 3RD THROUGH NTH CHARACTER */
  {COMPLEMENT TRANSMIT$TOGGLE}
END;
END;

END AUTO$REPEAT;

/* PROCEDURE TRANSMIT- ONCE THE HOST COMPUTER SIGNALS THE 8051H BY BRINGING
THE CLEAR-TO-SEND LINE LOW, THE ASCII CHARACTER IS PUT INTO THE SERIAL PORT.*/

TRANSMIT:
PROCEDURE;
IF {THE TERMINAL IS ON-LINE} THEN
DO;
  {WAIT UNTIL THE CLEAR$TO$SEND LINE IS LOW AND UNTIL THE 8051 SERIAL PORT TX IS NOT BUSY (TRANSMIT$INT=1)}
  {TRANSMIT THE ASCII CODE}
  {CLEAR THE FLAG "TRANSMIT$INT". THE SERIAL PORT SERVICE ROUTINE WILL SET THE FLAG
  WHEN THE SERIAL PORT IS FINISHED TRANSMITTING}
END;
ELSE {THE TERMINAL IS IN THE LOCAL MODE}
DO;
  {PUT THE ASCII CODE IN THE FIFO}
  {INCREMENT THE FIFO POINTER}
  {SET SERIAL$INT}
END;
END TRANSMIT;

```

```

/*      PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER'S MESSAGES AND DETERMINES
        WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE
        THE PROCEDURE THEN ACTS ACCORDINGLY */

```

```

DECIPHER:
START$DECIPHER:

```

```

VALID$RECEPTION=0;
DO WHILE {THE FIFO IS NOT EMPTY AND THE CHARACTER IS DISPLAYABLE}
    RECEIVE={ASCII CODE}
    CALL DISPLAY;
    {NEXT CHARACTER}
END;

```

```

IF {CHARACTERS WERE DISPLAYED} THEN
    {DISABLE SERIAL PORT INTERRUPT}
    {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
    {ENABLE SERIAL PORT INTERRUPT}
    {SET THE VALID$RECEPTION FLAG}

```

```

IF {THE FIFO IS EMPTY} THEN {CLEAR THE "SERIAL$INT FLAG AND RETURN}

```

```

IF {THE NEXT CHARACTER IS AN "ESC" CODE } THEN
DO:

```

```

    {LOOK AT THE CHARACTER IN THE FIFO AFTER THE ESC CODE AND CALL THE CORRECT SUBROUTINE}

```

```

        ;
        CALL UP$CURSER;                /* ESC A */
        CALL DOWN$CURSER;              /* ESC B */
        CALL RIGHT$CURSER;             /* ESC C */
        CALL LEFT$CURSER;              /* ESC D */
        CALL CLEAR$SCREEN;              /* ESC E */
        CALL MOV$CURSER;                /* ESC F */
        ;
        CALL HOME;                     /* ESC H */
        ;
        CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN; /* ESC J */
        CALL BLINK;                     /* ESC K */

```

```

    {DISABLE THE SERIAL PORT INTERRUPT}
    {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
    {ENABLE THE SERIAL PORT INTERRUPT}
    {SET THE "VALID$RECEPTION" FLAG}

```

```

IF {THE FIFO IS EMPTY} THEN {CLEAR THE SERIAL$INT FLAG AND RETURN}
END;

```

```

IF {THE NEXT CHARACTER IS A CONTROL CODE} THEN
DO;
  {CALL THE RIGHT SUBROUTINE}

      CALL LEFT$CURSER;          /* CTL H */
      ;
      CALL LINE$FEED;           /* CTL J */
      ;
      CALL CLEAR$SCREEN;        /* CTL L */
      CALL CARRIAGE$RETURN;     /* CTL M */

      {DISABLE THE SERIAL PORT INTERRUPT}
      {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
      {ENABLE THE SERIAL PORT INTERRUPT}
      {SET THE "VALID$RECEPTION" FLAG}
END;

IF {NO VALID CODE WAS RECEIVED ("VALID$RECEPTION" IS 0)} THEN
  {THROW THE CHARACTER OUT AND MOVE THE REMAINING CONTENTS OF THE FIFO}
  {UP TO THE BEGINNING}

IF {THE FIFO IS EMPTY} THEN {CLEAR THE SERIAL$INT FLAG AND RETURN}

END DECIPHER;

/*      PROCEDURE DISPLAY: THIS PROCEDURE WILL TAKE THE BYTE IN RAM LABELED
      RECEIVE AND PUT IT INTO THE DISPLAY RAM. */

DISPLAY:

  {PUT INTO THE DISPLAY RAM LOCATION POINTED TO BY "DISPLAY$RAM$POINTER
  THE CONTENTS OF RECEIVE}

  IF {THE END OF THE DISPLAY MEMORY HAS BEEN REACHED} THEN
    {RESET "DISPLAY$RAM$POINTER" TO THE BEGINNING OF THE RAM}
  ELSE
    {INCREMENT "DISPLAY$RAM$POINTER"}

  IF {THE CURSER IS IN THE LAST COLUMN OF THE CRT DISPLAY} THEN
  DO;
    {MOVE THE CURSER BACK TO THE BEGINNING OF THE LINE}
    IF {THE NEW DISPLAY RAM LOCATION HAS A END-OF-LINE CHARACTER IN IT} THEN
      CALL FILL;

    IF {THE CURSER IS ON THE LAST LINE OF THE CRT DISPLAY} THEN
      CALL SCROLL;
    ELSE
      {MOVE THE CURSER TO THE NEXT LINE}
  END;
  ELSE
    {INCREMENT THE CURSER TO THE NEXT LOCATION}

  {TURN THE CURSER ON }
  CALL LOADCURSER;
END DISPLAY;

```

```
/*      PROCEDURE LINE$FEED      */
```

```
LINE$FEED:
```

```
IF {THE CURSER IS IN THE LAST LINE OF THE CRT DISPLAY} THEN  
  CALL SCROLL;
```

```
ELSE
```

```
DO;
```

```
  {MOVE THE CURSER TO THE NEXT LINE}
```

```
  {TURN THE CURSER ON}
```

```
  CALL LOAD$CURSER;
```

```
END;
```

```
IF {THE DISPLAY$RAM$POINTER IS ON THE LAST LINE IN THE DISPLAY RAM} THEN
```

```
  {MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY RAM}
```

```
ELSE
```

```
  {MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY RAM}
```

```
IF {THE FIRST CHARACTER IN THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER } THEN
```

```
  CALL FILL;
```

```
END LINE$FEED;
```

```
/*      PROCEDURE SCROLL      */
```

```
SCROLL:
```

```
CALL BLANK;
```

```
{DISABLE VERTICAL RETRACE INTERRUPT}
```

```
IF {THE FIRST LINE OF THE CRT CONTAINS THE LAST LINE OF THE DISPLAY MEMORY} THEN
```

```
  {MOVE THE POINTER "LINE0" TO THE BEGINNING OF THE DISPLAY MEMORY}
```

```
ELSE
```

```
  {MOVE "LINE0" TO THE NEXT LINE IN THE DISPLAY MEMORY}
```

```
{ENABLE VERTICAL RETRACE INTERRUPT}
```

```
END SCROLL;
```

```
/*      PROCEDURE CLEAR SCREEN      */
```

```
CLEAR$SCREEN:
```

```
CALL HOME;
```

```
CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
```

```
END CLEAR$SCREEN;
```



```
/*      PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSER TO THE 0,0 POSITION */
```

```
HOME:
```

```
{MOVE THE CURSER POSITION TO THE UPPER LEFT HAND CORNER OF THE CRT}
{TURN THE CURSER ON}
CALL LOAD$CURSER;
{MOVE THE DISPLAY$RAM$POINTER TO THE CORRECT LOCATION IN THE DISPLAY RAM}
```

```
END HOME;
```

```
/*      PROCEDURE ERASE FROM CURSER TO END OF SCREEN:  */
```

```
ERASE$FROM$CURSER$TO$END$OF$SCREEN:
```

```
CALL BLINE;                                /* ERASE CURRENT LINE */
```

```
IF {THE CURSER IS NOT ON THE LAST LINE OF THE CRT DISPLAY} THEN
    STARTING WITH THE NEXT LINE,PUT AN END-OF-LINE CHARACTER (0F1H)
    IN THE DISPLAY RAM LOCATIONS THAT CORRESPOND TO THE BEGINNING OF
    THE CRT DISPLAY LINES UNTIL THE BOTTOM OF THE CRT SCREEN HAS BEEN REACHED}
END;
```

```
END ERASE$FROM$CURSER$TO$END$OF$SCREEN;
```

```
/*PROCEDURE MOV$CURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR
IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL
READ THE NEXT TWO BYTE TO DETERMINE WHERE TO MOVE THE CURSER. THE FIRST BYTE
IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION */
```

```
MOV$CURSER:
```

```
{WAIT UNTIL THE FIFO HAS RECEIVED THE NEXT TWO CHARACTERS}
{MOVE THE CURSER TO THE LOCATION SPECIFIED IN THE ESCAPE SEQUENCE}
{MOVE THE DISPLAY$RAM$POINTER TO THE CORRECT LOCATION}
```

```
IF THE FIRST CHARACTER IN THE NEW LINE HAS AN END-OF-LINE CHARACTER} THEN
    CALL FILL;
END;
```

```
{DISABLE THE SERIAL PORT INTERRUPT}
{MOVE THE REMAIN CONTENTS OF THE FIFO UP TWO LOCATIONS IN MEMORY}
{DECREMENT THE FIFO BY TWO}
{ENABLE THE SERIAL PORT INTERRUPT}
```

```
END MOV$CURSER;
```

```
/*      PROCEDURE LEFT CURSER: THIS PROCEDURE MOVES THE CURSER LEFT ONE COLUMN
BY SUBTRACTING 1 OF THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */
```

```
LEFT$CURSER:
```

```
IF {THE CURSER IS NOT IN THE FIRST LOCATION OF A LINE} THEN
DO;
```

```
{MOVE THE CURSER LEFT BY ONE LOCATION}
{TURN THE CURSER ON}
CALL LOAD$CURSER;
{DECREMENT THE DISPLAY$RAM$POINTER BY ONE}
```

```
END;
```

```
END LEFT$CURSER;
```

```
/*      PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */
```

```
RIGHT$CURSER:
```

```
IF {THE CURSER IS NOT IN THE LAST POSITION OF THE CRT LINE} THEN
DO;
  {MOVE THE CURSER RIGHT BY ONE LOCATION}
  {TURN THE CURSER ON}
  CALL LOAD$CURSER;
  {INCREMENT THE DISPLAY$RAM$POINTER BY ONE}
END;

END RIGHT$CURSER;
```

```
/*      PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */
```

```
UP$CURSER:
```

```
IF {THE CURSER IS NOT ON THE FIRST LINE OF THE CRT DISPLAY} THEN
DO;
  {MOVE THE CURSER UP ONE LINE}
  {TURN ON THE CURSER}
  CALL LOAD$CURSER;

  IF {THE DISPLAY$RAM$POINTER IS IN THE FIRST LINE OF DISPLAY MEMORY} THEN
    {MOVE THE DISPLAY$RAM$POINTER TO THE LAST LINE OF DISPLAY MEMORY}
  ELSE
    {MOVE THE DISPLAY$RAM$POINTER UP ONE LINE IN DISPLAY MEMORY}

  IF {THE FIRST LOCATION OF THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER} THEN
    CALL FILL;
END;

END UP$CURSER;
```

```
/*      PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */
```

```
DOWN$CURSER:
```

```
IF {THE CURSER IS NOT ON THE LAST LINE OF THE CRT DISPLAY} THEN
DO;
  {TURN THE CURSER ON}
  {MOVE THE CURSER TO THE NEXT LINE}
  CALL LOAD$CURSER;

  IF {THE DISPLAY$RAM$POINTER IS NOT ON THE LAST LINE OF THE DISPLAY MEMORY} THEN
    {MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY MEMORY}
  ELSE
    {MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY MEMORY}

  IF {THE FIRST CHARACTER IN THE NEW LINE IS AN END-OF-LINE CHARACTER} THEN
    CALL FILL;
END;

END DOWN$CURSER;
```

```

/*      PROCEDURE CARRIAGE$RETURN      */

CARRIAGE$RETURN:
{MOVE THE DISPLAY$RAM$POINTER TO THE BEGINNING OF THE CURRENT LINE IN THE DISPLAY MEMORY}
{MOVE THE CURSER TO THE BEGINNING OF THE CURRENT LINE OF THE CRT DISPLAY}
{TURN THE CURSER ON}
CALL LOAD$CURSER;

END CARRIAGE$RETURN;

/*      PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND
LOADS IT INTO THE 8276 CURSER REGISTER. */

LOAD$CURSER:
PROCEDURE;
IF {THE CURSER IS ON} THEN
    {MOVE THE CURSER BACK ONTO THE CRT DISPLAY}
{DISABLE BUFFER INTERRUPT}
{WRITE TO THE 8276 CURSER REGISTERS THE X,Y LOCATIONS}
{ENABLE BUFFER INTERRUPT}

END LOAD$CURSER;

/*      PROCEDURE CHECK BAUD RATE: THIS PROCEDURE READS THE THREE PORT PINS ON P1 AND SETS UP
THE SERIAL PORT FOR THE SPECIFIED BAUD RATE */

CHECK$BAUD$RATE:
{SET TIMER 1 TO MODE 1 AND AUTO RELOAD}
{TURN TIMER ON}
{ENABLE SERIAL PORT INTERRUPT}
{READ BAUD RATE SWITCHES AND SET UP RELOAD VALUE}

;          /* 00 IS NOT ALLOWED */
TH1=040H;    /* 150 BAUD */
TH1=0A0H;    /* 300 BAUD */
TH1=0D0H;    /* 600 BAUD */
TH1=0E8H;    /* 1200 BAUD */
TH1=0F4H;    /* 2400 BAUD */
TH1=0FAH;    /* 4800 BAUD */
TH1=0FDH;    /* 9600 BAUD */

END CHECK$BAUD$RATE;

```

```

/*      PROCEDURE READER: THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. THE
EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN
LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL
MEMORY ARRAY CALLED CURRENT$KEY */

READER:

{ INITIALIZE FLAGS "KEY0"=0, "SAME"=1, 0 COUNTER=0}

DO UNTIL {ALL 8 KEYBOARD SCAN LINES ARE READ}
  {READ KEYBOARD SCAN}
  IF {NO KEY WAS PRESSED} THEN
    {INCREMENT 0 COUNTER}
  ELSE
    IF {THE KEY PRESSED WAS NOT THE SAME KEY THAT WAS PRESSED THE LAST TIME
      THE KEYBOARD WAS READ} THEN
      {CLEAR "SAME" AND WRITE NEW SCAN RESULT TO CURRENT$KEY RAM ARRAY}
    END;
  IF {ALL 8 SCANS DIDN'T HAVE A KEY PRESSED (0 COUNTER=8)} THEN
    {SET KEY0, AND CLEAR SAME}
  END;
END READER;

/*      PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 WITH SPACES (20H ASCII)
DURING THE SCROLL ROUTINES.*/

BLANK:

DO I= {BEGINNING OF THE CRT DISPLAY (LINE0)} TO {LINE0 + 50H}
  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
  NEXT I
END;

END BLANK;

/*      PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSER TO THE END OF
THE DISPLAY LINE */

BLINE:

DO I= {CURRENT CURSER POSITION ON CRT DISPLAY} TO {END OF ROW}
  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
  NEXT I
END;

END BLINE;

/*      PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS A DISPLAY LINE WITH SPACES*/

FILL:

DO I= {BEGINNING OF THE LINE THAT THE CURSER IS ON} TO {END OF THE ROW}
  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
  NEXT I
END;

END FILL;

```


PL/M-51 COMPILER

```

                $EJECT

                CRT$CONTROLLER:
1   1   DO;

                /***** DECLARE LITERALS *****/

2   1   DECLARE LLC LITERALLY 'LOCAL$LINE$CHANGE';
3   1   DECLARE REG LITERALLY 'REGISTER';
4   1   DECLARE CURRENT$KEY LITERALLY 'CURKEY';
5   1   DECLARE SERIAL$SERVICE LITERALLY 'SERBUF';
6   1   DECLARE DISPLAY$RAM$POINTER LITERALLY 'POINT';
7   1   DECLARE SERIAL$INT LITERALLY 'SERINT';
8   1   DECLARE TRANSMIT$INT LITERALLY 'TRNINT';
9   1   DECLARE CURSER$COLUMN LITERALLY 'CURSER';
10  1   DECLARE LAST$KEY LITERALLY 'LSTKEY';
11  1   DECLARE CURSER$COUNT LITERALLY 'COUNT';
12  1   DECLARE SCAN$INT LITERALLY 'SCAN';

                /***** REGISTER DECLARATIONS FOR THE 8051 *****/

                /***** BYTE REGISTERS *****/

13  1   DECLARE
        P0 BYTE AT(80H) REG,
        P1 BYTE AT(90H) REG,
        P2 BYTE AT(0A0H) REG,
        P3 BYTE AT(0B0H) REG,
        PSW BYTE AT(0D0H) REG,
        ACC BYTE AT(0E0H) REG,
        B BYTE AT(0F0H) REG,
        SP BYTE AT(81H) REG,
        DPL BYTE AT(82H) REG,
        DPH BYTE AT(83H) REG,
        PCON BYTE AT(87H) REG,
        TCON BYTE AT(88H) REG,
        TMOD BYTE AT(89H) REG,
        TL0 BYTE AT(8AH) REG,
        TL1 BYTE AT(8BH) REG,
        TH0 BYTE AT(8CH) REG,
        TH1 BYTE AT(8DH) REG,
        IE BYTE AT(0A8H) REG,
        IP BYTE AT(0B8H) REG,
        SCON BYTE AT(98H) REG,
        SBUF BYTE AT(99H) REG;

```

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT
/***** BIT REGISTERS *****/

/***** PSW BITS *****/
14 1 DECLARE
    CY BIT AT (0D7H) REG,
    AC BIT AT (0D6H) REG,
    F0 BIT AT (0D5H) REG,
    RS1 BIT AT (0D4H) REG,
    RS0 BIT AT (0D3H) REG,
    OV BIT AT (0D2H) REG,
    P BIT AT (0D0H) REG,

/***** TCON BITS *****/
    TF1 BIT AT (8FH) REG,
    TR1 BIT AT (8EH) REG,
    TF0 BIT AT (8DH) REG,
    TR0 BIT AT (8CH) REG,
    IE1 BIT AT (8BH) REG,
    IT1 BIT AT (8AH) REG,
    IE0 BIT AT (89H) REG,
    IT0 BIT AT (88H) REG,

/***** IE BITS *****/
    EA BIT AT (0AFH) REG,
    ES BIT AT (0ACH) REG,
    ET1 BIT AT (0ABH) REG,
    EX1 BIT AT (0AAH) REG,
    ET0 BIT AT (0A9H) REG,
    EX0 BIT AT (0A8H) REG,

/***** IP BITS *****/
    PS BIT AT (0BCH) REG,
    PT1 BIT AT (0BBH) REG,
    PK1 BIT AT (0BAH) REG,
    PT0 BIT AT (0B9H) REG,
    PK0 BIT AT (0B8H) REG,

/***** P3 BITS *****/
    RD BIT AT (0B7H) REG,
    WR BIT AT (0B6H) REG,
    T1 BIT AT (0B5H) REG,
    T0 BIT AT (0B4H) REG,
    INT1 BIT AT (0B3H) REG,
    INT0 BIT AT (0B2H) REG,
    TXD BIT AT (0B1H) REG,
    RXD BIT AT (0B0H) REG,

/***** SCON BITS *****/
    SM0 BIT AT (9FH) REG,
    SM1 BIT AT (9EH) REG,
    SM2 BIT AT (9DH) REG,
    REN BIT AT (9CH) REG,
    TB8 BIT AT (9BH) REG,
    RB8 BIT AT (9AH) REG,
    TI BIT AT (99H) REG,
    RI BIT AT (98H) REG;

```

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT
$IF SW1
/***** DECLARE CONSTANTS*****/
15 1  DECLARE LOW$SCAN(16) STRUCTURE
      (KEY (8) BYTE) CONSTANT

      ('890-',5CH,5EH,08H,00H,
/* SCAN 0, SHIFT KEY =0; 8,9,0,-,\,^, BACK SPACE */

      'uiop',5BH,'@',0AH,7FH,
/* SCAN 1, SHIFT =0; u,i,o,p,[,], LINE FEED, DELETE */

      'jkl;',00H,0DH,'7',
/* SCAN 2, SHIFT =0; j,k,l,;,:, RETURN, 7 */

      'm',2CH,'.',00H,'/',00H,00H,00H,
/* SCAN 3, SHIFT =0; m,COMMA,./ */

      00H,'azxcvbn',
/* SCAN 4, SHIFT =0; a,z,x,c,v,b,n */

      'y',00H,00H,' d f g h',
/* SCAN 5, SHIFT =0; y, SPACE, d,f,g,h */

      09H,'qwsert',00H,
/* SCAN 6, SHIFT =0; TAB,q,w,s,e,r,t */

      1BH,'123456',00H,
/* SCAN 7, SHIFT =0;ESC,1,2,3,4,5,6 */

      28H,29H,00H,'=',7CH,7EH,08H,00H,
/* SCAN 0, SHIFT =1; (,),=,|,~, BACK SPACE */

      'UIOP',00H,00H,0AH,7FH,
/* SCAN 1, SHIFT =1; U,I,O,P, LINE FEED, DELETE */

      'JKL+*',00H,0DH,27H,
/* SCAN 2, SHIFT =1; J,K,L,+,*, RETURN, ' */

      'M<>',00H,3FH,00H,00H,00H,
/* SCAN 3, SHIFT =1; M,<,>,* */

      00H,'AZXCVEN',
/* SCAN 4, SHIFT =1; A,Z,X,C,V,B,N */

      'Y',00H,00H,' D F G H',
/* SCAN 5, SHIFT =1; Y, SPACE, D,F,G,H */

      09H,'QWERT',00H,
/* SCAN 6, SHIFT =1; TAB, Q,W,S,E,R,T */

      1BH,'!""$%&',00H);
/* SCAN 7, SHIFT =1;ESC,!,"#$%& */
$ENDIF

```


PL/M-51 COMPILER CRIOCONTROLLER

```

$EJECT
/*****DECLARE VARIABLES*****/

16 1  DECLARE

    $IF SW2

INPUT      BIT AT (0B4H)    REG,
$ENDIF
$IF SW1
    CAP$LOCK      BIT AT (095H)    REG,
    SHIFT$KEY     BIT AT (096H)    REG,
    CONTROL$KEY   BIT AT (097H)    REG,
$ENDIF
    LOCAL$LINE    BIT AT (0B5H)    REG,
    CLEAR$TO$SEND BIT AT (093H)    REG,
    DATA$TERMINAL$READY BIT AT (094H)    REG;

17 1  DECLARE (

    $IF SW1
        SAME,
        VALID$KEY,
        KEY 0,
        LAST$SHIFT$KEY,
        LAST$CONTROL$KEY,
        LAST$CAP$LOCK,
    $ENDIF

    $IF SW2
        RCVFLG,
        SYNC,
        BYFIN,
        KBDINT,
        ERROR,
    $ENDIF

        NEWSKEY,
        TRANSMIT$TOGGLE,
        CURSER$ON,
        SERIAL$INT,
        SCAN$INT,
        TRANSMIT$INT,
        ESCSEQ,
        VALID$RECEPTION,
        LLC,
        ENSP)      BIT PUBLIC;

```

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT

18 1  DECLARE (
        I,
        J,
        K,
        ASCII$KEY,
        TRANSMIT$COUNT,
        TEMP,
        SHIFT,
        CURSER$COL,
        CURSER$COLUMN,
        CURSER$ROW,
        CURSER$COUNT,
        FIFO,
        RECEIVE)      BYTE PUBLIC;

$IF SW1
19 1  DECLARE LAST$KEY (8) BYTE PUBLIC;
$ENDIF

$IF SW2
    DECLARE LAST$KEY (2) BYTE PUBLIC;
$ENDIF

20 1  DECLARE SERIAL (16)  BYTE PUBLIC;

21 1  DECLARE DISPLAY$RAM (7CFH) BYTE AT (1000H)  AUXILIARY;

22 1  DECLARE
        PARAMETERS$ADDRESS  BYTE AT (0000H)  AUXILIARY,
        COMMAND$ADDRESS     BYTE AT (0001H)  AUXILIARY;

23 1  DECLARE (
        DISPLAY$RAM$POINTER,
        RASTER,
        LINE0,
        L)      WORD PUBLIC;

```

PL/M-51 COMPILER CRYPTOCONTROLLER

\$EJECT

```

/* PROCEDURE READER: THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. THE
EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN
LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL
MEMORY ARRAY CALLED CURRENTSKEY. THE PROCEDURE CONTROLS 2 STATUS FLAGS;
KEY0 AND SAME. KEY0 IS SET IF ALL 8 SCANS READ NO KEY WAS PRESSED.
IF ALL 8 SCANS ARE THE SAME AS THE LAST READING OF THE KEYBOARD, THEN
SAME IS SET. */

```

```

24 2 READER: PROCEDURE EXTERNAL;
25 1 END READER;

```

```

/* PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 SCAN WITH SPACES (20H ASCII)
DURING THE SCROLL ROUTINES.*/

```

```

26 2 BLANK: PROCEDURE EXTERNAL;
27 1 END BLANK;

```

```

/* PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSER TO THE END OF
THE DISPLAY LINE */

```

```

28 2 BLINE: PROCEDURE EXTERNAL;
29 1 END BLINE;

```

```

/* PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS THE CURSER LINE
WITH SPACES*/

```

```

30 1 FILL:
PROCEDURE EXTERNAL;
31 1 END FILL;

```

PL/M-51 COMPILER CRICONTROLLER

```

$EJECT

/* PROCEDURE CHECK BAUD RATE: THIS PROCEDURE READS THE THREE PORT PINS ON P1 AND SETS UP
THE SERIAL PORT FOR THE SPECIFIED BAUD RATE */

32 1 CHECK$BAUD$RATE:
PROCEDURE;
33 2 SCON=70H; /* MODE 1
/* ENABLE RECEPTION*/
34 2 TMOD=TMOD OR 20H; /* TIMER 1 AUTO RELOAD */
35 2 TR1=1; /* TIMER 1 ON */
36 2 ES=1; /* ENABLE SERIAL INTERRUPT*/
37 2 ENSP=1; /* SERIAL INTERRUPT MASK FLAG */
38 3 DO CASE (P1 AND 07H);
39 3 ; /* 00 IS NOT ALLOWED */
40 3 TH1=040H; /* 150 BAUD */
41 3 TH1=0A0H; /* 300 BAUD */
42 3 TH1=0D0H; /* 600 BAUD */
43 3 TH1=0E8H; /* 1200 BAUD */
44 3 TH1=0F4H; /* 2400 BAUD */
45 3 TH1=0FAH; /* 4800 BAUD */
46 3 TH1=0FDH; /* 9600 BAUD */
47 3 END;
48 1 END CHECK$BAUD$RATE;

/* PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND
LOADS IT INTO THE 8276 CURSER REGISTERS. */

49 1 LOAD$CURSER:
PROCEDURE;
50 2 IF CURSER$ON=1 THEN
51 2 CURSER$COL=CURSER$COLUMN;
52 2 EX1=0; /* DISABLE BUFFER INTERRUPT */
53 2 COMMAND$ADDRESS=80H; /* INITIALIZE CURSER COMMAND */
54 2 PARAMETER$ADDRESS=CURSER$COL;
55 2 PARAMETER$ADDRESS=CURSER$ROW;
56 2 EX1=1; /* ENABLE BUFFER INTERRUPT */
57 1 END LOAD$CURSER;

/* PROCEDURE CARRIAGE$RETURN */

58 1 CARRIAGE$RETURN:
PROCEDURE;
59 2 DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-CURSER$COLUMN;
60 2 CURSER$COLUMN=0;
61 2 CURSER$ON=1;
62 2 CALL LOAD$CURSER;
63 1 END CARRIAGE$RETURN;

```

PL/M-51 COMPILER CRTCONTROLLER

```

SELECT

/* PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

64 1 DOWN$CURSER:
PROCEDURE;
65 2 IF CURSER$ROW < 18H THEN
66 3 DO;
67 3 CURSER$ON=1;
68 3 CURSER$ROW=CURSER$ROW + 1;
69 3 CALL LOAD$CURSER;
70 3 IF DISPLAY$RAM$POINTER < 780H THEN
71 3 DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER + 50H;
72 3 ELSE
73 3 DISPLAY$RAM$POINTER=(DISPLAY$RAM$POINTER-780H);
74 3 L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
75 3 IF DISPLAY$RAM(L)=0F1H THEN /* LOOK FOR END OF*/
76 3 DO; /* LINE CHARACTER */
77 3 CALL FILL; /*IF TRUE FILL LINE*/
78 3 DISPLAY$RAM(L)=20H; /*WITH SPACES */
79 3 END;
80 1 END DOWN$CURSER;

/* PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

81 1 UP$CURSER:
PROCEDURE;
82 2 IF CURSER$ROW > 0 THEN
83 3 DO;
84 3 CURSER$ROW=CURSER$ROW - 1;
85 3 CURSER$ON=1;
86 3 CALL LOAD$CURSER;
87 3 IF DISPLAY$RAM$POINTER<50H THEN
88 3 DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+780H;
89 3 ELSE
90 3 DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER - 50H;
91 3 L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
92 3 IF DISPLAY$RAM(L)=0F1H THEN /* LOOK FOR END OF LINE*/
93 3 DO; /* CHARACTER */
94 3 CALL FILL; /* IF TRUE FILL WITH */
95 3 DISPLAY$RAM(L)=20H; /* SPACES */
96 3 END;
97 1 END UP$CURSER;

```

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT

/* PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
  BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

98  1  RIGHT$CURSER:
    PROCEDURE;
99  2  IF CURSER$COLUMN < 4FH THEN
100  3  DO;
101  3      CURSER$COLUMN=CURSER$COLUMN + 1;
102  3      CURSER$ON=1;
103  3      CALL LOAD$CURSER;
104  3      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER +1;
105  3  END;
106  1  END RIGHT$CURSER;

/* PROCEDURE LEFT CURSER: THIS PROCEDURE MOVES THE CURSER LEFT ONE COLUMN
  BY SUBTRACTING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

107  1  LEFT$CURSER:
    PROCEDURE;
108  2  IF CURSER$COLUMN >0 THEN
109  3  DO;
110  3      CURSER$COLUMN=CURSER$COLUMN - 1;
111  3      CURSER$ON=1;
112  3      CALL LOAD$CURSER;
113  3      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER -1;
114  3  END;
115  1  END LEFT$CURSER;

```

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT

/* PROCEDURE MOV$CURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR
IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL
READ THE NEXT TWO BYTE TO DETERMINE WHERE TO MOVE THE CURSER. THE FIRST BYTE
IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION */

116 1  MOV$CURSER:
117 3  PROCEDURE;
118 3  DO WHILE FIFO<4; /* WAIT UNTILL THE MOV$CURSER PARAMETERS*/
119 3  END; /* ARE RECEIVED INTO THE FIFO */
120 2  TEMP=CURSER$ROW;
121 2  CURSER$ROW=SERIAL(2);
122 2  IF CURSER$ROW>TEMP THEN
123 3  DO;
124 3  L=DISPLAY$RAM$POINTER+ ((CURSER$ROW-TEMP)*50H);
125 3  IF L>7CFH THEN /* IF OUT OF RAM RANGE */
126 3  ELSE DISPLAY$RAM$POINTER=L-7D0H; /* RAP AROUND TO BEGINNING */
127 3  ELSE DISPLAY$RAM$POINTER=L; /* OF RAM */
128 2  END;
129 3  IF CURSER$ROW<TEMP THEN
130 4  DO;
131 4  L= (TEMP-CURSER$ROW)*50H;
132 4  IF DISPLAY$RAM$POINTER<L THEN /* IF OUT OF RAM RANGE*/
133 4  DISPLAY$RAM$POINTER= (7D0H- (L-DISPLAY$RAM$POINTER)); /* RAP AROUND TO END OF RAM*/
134 4  ELSE
135 4  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-L;
136 3  END;
137 2  TEMP=CURSER$COLUMN;
138 2  CURSER$COLUMN=SERIAL(3);
139 2  IF CURSER$COLUMN>TEMP THEN
140 3  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+ (CURSER$COLUMN-TEMP);
141 2  ELSE
142 3  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER- (TEMP-CURSER$COLUMN);
143 2  CURSER$ON=1;
144 2  CALL LOAD$CURSER;
145 2  L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
146 2  IF DISPLAY$RAM(L)=0F1H THEN /* LOOK FOR END FO LINE CHARACTER*/
147 3  DO;
148 3  CALL FILL; /* IF TRUE FILL WITH SPACES */
149 3  DISPLAY$RAM(L)=20H;
150 2  END;
151 2  ES=0;
152 3  DO I=2 TO FIFO-2;
153 3  SERIAL(I)=SERIAL(I+2);
154 2  END;
155 2  FIFO=FIFO-2;
156 2  ES=ENSP;
157 1  END MOV$CURSER;

```

PL/M-51 COMPILER CR/CONTROLLER

```

        SEJECT

        /*  PROCEDURE ERASE FROM CURSER TO END OF SCREEN:  */

157  1    ERASE$FROM$CURSER$TO$END$OF$SCREEN:
        PROCEDURE;
158  2    CALL BLINE;                                /* ERASE CURRENT LINE */
159  2    IF CURSER$ROW < 18H THEN
160  3    DO;
161  3        L=DISPLAY$RAM$POINTER-CURSER$COLUMN+50H;    /* GET NEXT LINE */
162  4        DO WHILE (L < 7D0H) AND (L <> (LINE0 AND 7FFH));
163  4            DISPLAY$RAM(L)=0F1H;                    /* ERASE UNTIL LINE0 OR */
164  4            L=L+50H;                                /* END OF DISPLAY RAM*/
165  4        END;
166  3        L=0;
167  4        DO WHILE L <> (LINE0 AND 7FFH);            /* ERASE UNTIL LINE0 */
168  4            DISPLAY$RAM(L)=0F1H;
169  4            L=L+50H;
170  4        END;
171  3    END;
172  1    END ERASE$FROM$CURSER$TO$END$OF$SCREEN;

        /*  PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSER TO THE 0,0 POSITION */

173  1    HOME:
        PROCEDURE;
174  2    CURSER$ROW=00;
175  2    CURSER$COLUMN=00;
176  2    CURSER$ON=1;
177  2    CALL LOAD$CURSER;
178  2    DISPLAY$RAM$POINTER=(LINE0 AND 7FFH);
179  1    END HOME;

```


PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT

/*  PROCEDURE CLEAR SCREEN  */

180 1  CLEAR$SCREEN:
      PROCEDURE;
181 2  CALL HOME;
182 2  CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
183 1  END CLEAR$SCREEN;

/*  PROCEDURE SCROLL  */

184 1  SCROLL:
      PROCEDURE;
185 2  CALL BLANK;
186 2  EX0=0; /* DISABLE VERTICAL REFRESH INTERRUPT */
187 2  IF LINE0= 1F80H THEN
188 2  LINE0= 1800H;
189 2  ELSE
      LINE0= LINE0+50H;
190 2  EX0=1; /* ENABLE VERTICAL REFRESH INTERRUPT */
191 1  END SCROLL;

/*  PROCEDURE LINE$FEED  */

192 1  LINE$FEED:
      PROCEDURE;
193 2  IF CURSER$ROW=18H THEN
194 2  CALL SCROLL;
195 2  ELSE
      DO;
196 3  CURSER$ROW= CURSER$ROW+1;
197 3  CURSER$ON=1;
198 3  CALL LOAD$CURSER;
199 3  END;
200 2  IF DISPLAY$RAM$POINTER > 77FH THEN
201 2  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-780H;
202 2  ELSE
      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+50H;
203 2  L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
204 2  IF DISPLAY$RAM(L)=0F1H THEN /* LOOK FOR END OF LINE CHARACTER */
205 3  DO;
206 3  CALL FILL; /* IF TRUE FILL WITH SPACES */
207 3  DISPLAY$RAM(L)=20H;
208 3  END;
209 1  END LINE$FEED;

```

PL/M-51 COMPILER CRYPTOCONTROLLER

```

$EJECT

/*  PROCEDURE DISPLAY: THIS PROCEDURE WILL TAKE THE BYTE IN RAM LABELED
    RECEIVE AND PUT IT INTO THE DISPLAY RAM. */

210  1  DISPLAY :
211  2  PROCEDURE;
212  2  DISPLAY$RAM(DISPLAY$RAM$POINTER)=RECEIVE;
213  2  IF DISPLAY$RAM$POINTER=7CFH THEN          /* IF END OF RAM */
214  2  DISPLAY$RAM$POINTER=000H;                /* RAP AROUND TO BEGINNING */
215  2  ELSE
216  3  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+1;
217  3  IF CURSER$COLUMN=4FH THEN
218  4  DO;
219  5  CURSER$COLUMN=00H;
220  5  L=DISPLAY$RAM$POINTER;
221  5  IF DISPLAY$RAM(L)=0F1H THEN
222  6  DO;
223  7  CALL FILL;
224  7  DISPLAY$RAM(L)=20H;
225  7  END;
226  6  IF CURSER$ROW=18H THEN
227  7  CALL SCROLL;
228  6  ELSE
229  7  CURSER$ROW=CURSER$ROW+1;
230  7  END;
231  5  CURSER$COLUMN=CURSER$COLUMN+1;
232  5  CURSER$ON=1;
233  5  CALL LOADCURSER;
234  5  END DISPLAY;

```

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT

/* PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER'S MESSAGES AND DETERMINES
   WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE
   THE PROCEDURE THEN ACTS ACCORDINGLY */

232 1  DECIPHER:
233 2  PROCEDURE;
234 2  START$DECIPHER;
235 3  VALID$RECEPTION=0;
236 3  I=0;
237 3  DO WHILE (I<FIFO) AND (SERIAL(I)>1FH) AND (SERIAL(I)<7FH);
238 3  RECEIVE=SERIAL(I);
239 3  CALL DISPLAY;
240 3  I=I+1;
241 3  END;
242 2  IF I>0 THEN
243 3  DO;
244 3  ES=0; /* DISABLE SERIAL INTERRUPT WHILE MOVING FIFO */
245 3  K=FIFO-I;
246 3  DO J=0 TO K; /* MOVE FIFO */
247 3  SERIAL(J)=SERIAL(I);
248 3  I=I+1;
249 3  END;
250 3  FIFO=K;
251 3  ES=ENSP; /* ENABLE SERIAL INTERRUPT */
252 3  VALID$RECEPTION=1;
253 3  END;
254 2  IF FIFO=0 THEN
255 3  DO;
256 3  SERIAL$INT=0;
257 3  GOTO END$DECIPHER;
258 3  END;
259 2  IF (SERIAL(0)=1BH) THEN
260 3  DO;
261 3  IF (ESC$SEQ=1) AND (FIFO<2) THEN
262 3  GOTO END$DECIPHER;
263 3  K=(SERIAL(1) AND 5FH)-40H;
264 3  IF (K > 00H) AND (K < 0CH) THEN
265 4  DO;
266 5  DO CASE K;
267 5  ;
268 5  CALL UP$CURSER; /* ESC A */
269 5  CALL DOWN$CURSER; /* ESC B */
270 5  CALL RIGHT$CURSER; /* ESC C */
271 5  CALL LEFT$CURSER; /* ESC D */
272 5  CALL CLEAR$SCREEN; /* ESC E */
273 5  CALL MOV$CURSER; /* ESC F */
274 5  ;
275 5  CALL HOME; /* ESC H */
276 5  ;
277 5  CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN; /* ESC J */
278 5  CALL BLINK; /* ESC K */
279 5  END;
280 3  END;
281 2  ES=0; /* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO */

```

PL/M-51 COMPILER CRTCONTROLLER

```

280 4      DO I=0 TO (FIFO-2);
281 4          SERIAL(I)=SERIAL(I+2);    /* MOVE FIFO */
282 4      END;
283 3      FIFO=FIFO-2;
284 3      ES=ENSP;                      /* ENABLE SERIAL INTERRUPTS */
285 3      VALID$RECEPTION=1;
286 3      IF FIFO=0 THEN
287 4          DO;
288 4              SERIAL$INT=0;
289 4              GOTO ENDS$DECIPHER;
290 4          END;
291 3      END;
292 2      IF (SERIAL(0)> 07H) AND (SERIAL(0)<0EH) THEN
293 3          DO;
294 4              DO CASE (SERIAL(0) -08H);
295 4                  CALL LEFT$CURSER;          /* CTL H */
296 4                  ;
297 4                  CALL LINE$FEED;            /* CTL J */
298 4                  ;
299 4                  CALL CLEAR$SCREEN;         /* CTL L */
300 4                  CALL CARRIAGE$RETURN;     /* CTL M */
301 4              END;
302 3              ES=0;                        /* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO */
303 4              DO I=0 TO (FIFO-1);
304 4                  SERIAL(I)=SERIAL(I+1);    /* MOVE FIFO */
305 4              END;
306 3              FIFO=FIFO-1;
307 3              ES=ENSP;                      /* ENABLE SERIAL INTERRUPTS */
308 3              VALID$RECEPTION=1;
309 3          END;
310 2      IF VALID$RECEPTION=0 THEN
311 3          DO;
312 3              ES=0;
313 4              DO I=0 TO (FIFO-1);          /* IF CHARACTER IS UNRECOGNIZED THEN */
314 4                  SERIAL(I)=SERIAL(I+1);    /* TRASH IT */
315 4              END;
316 3              FIFO=FIFO-1;
317 3              ES=ENSP;
318 3          END;
319 2      IF FIFO=0 THEN
320 2          SERIAL$INT=0;
321 2      ENDS$DECIPHER:
      END DECIPHER;

```

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT

/* PROCEDURE TRANSMIT- THIS PROCEDURE LOOKS AT THE CLEAR TO SEND PIN FOR AN ACTIVE
LOW SIGNAL. ONCE THE MAIN COMPUTER SIGNALS THE 8051 THE ASCII CHARACTER IS PUT
INTO THE SERIAL PORT.*/

322 1  TRANSMIT:
323 2  PROCEDURE;
324 3  IF LOCAL$LINE =1 THEN
325 4  DO WHILE (CLEAR$TO$SEND=1) OR (TRANSMIT$INT=0);
326 4  END;
327 3  SBUF=ASCII$KEY;
328 3  TRANSMIT$INT=0;
329 3  END;
330 2  ELSE
331 3  DO;
332 3  SERIAL(FIFO)=ASCII$KEY;
333 3  FIFO=FIFO+1;
334 3  SERIAL$INT=1;
335 1  END;
336 1  END TRANSMIT;

/* PROCEDURE AUTO$REPEAT: THIS PROCEDURE WILL PERFORM AN AUTO REPEAT FUNCTION
AFTER A FIXED DELAY PERIOD */

336 1  AUTO$REPEAT:
337 2  PROCEDURE;
338 3  IF NEWSKEY=1 THEN
339 3  DO;
340 3  TRANSMIT$TOGGLE=0;
341 3  TRANSMIT$COUNT=000H;
342 3  CALL TRANSMIT;
343 3  NEWSKEY=0;
344 2  ELSE
345 3  DO;
346 4  IF TRANSMIT$COUNT <> 00H THEN
347 4  DO;
348 4  TRANSMIT$COUNT=TRANSMIT$COUNT+1;
349 4  IF TRANSMIT$COUNT=0FFH THEN /*DELAY BETWEEN FIRST CHARACTER AND THE SECOND */
350 4  DO;
351 4  CALL TRANSMIT;
352 4  TRANSMIT$COUNT=00;
353 4  END;
354 3  ELSE
355 4  DO;
356 4  CURSER$ON=1;
357 4  CURSER$COUNT=0;
358 4  IF TRANSMIT$TOGGLE = 1 THEN /* 2 VERT FRAMES BETWEEN 3RD TO NTH CHARACTER */
359 4  CALL TRANSMIT; /* 3RD THROUGH NTH CHARACTER */
360 4  TRANSMIT$TOGGLE= NOT TRANSMIT$TOGGLE;
361 3  END;
362 1  END AUTO$REPEAT;

```

PL/M-51 COMPILER CRIOCONTROLLER

```

$EJECT

/***** START MAIN PROGRAM *****/
/* BEGIN BY PUTTING ASCII CODE FOR BLANK IN THE DISPLAY RAM;*/

363 1    INIT:
364 2    DO L=0 TO 7FH;
365 2    DISPLAY$RAM(L)=20H;
366 2    END;

/* INITIALIZE POINTERS, RAM BITS, ETC. */

366 1    ESC$SEQ=0;
367 1    SCAN$INT=0;
368 1    SERIAL$INT=0;
369 1    FIFO=0;
370 1    CURSER$COUNT=0;
371 1    LLC=0;
372 1    DATA$TERMINAL$READY=1;
373 1    TOON=05H;
374 1    LINE0=1800H;
375 1    RASTER=1800H;
376 1    DISPLAY$RAM$POINTER=0000H;
377 1    TRANSMIT$INT=1;

$IF SW1

378 2    DO I=0 TO 7;
379 2    LAST$KEY(I)=00H;
380 2    END;

381 1    VALID$KEY=0;
382 1    LAST$SHIFT$KEY=1;
383 1    LAST$CONTROL$KEY=1;
384 1    LAST$CAP$LOCK=1;
385 1    SENDIF

$IF SW2
RCVFLG=0;
SYNC=0;
BYFIN=0;
KBDINT=0;
ERROR=0;
SENDIF

/* INITIALIZE THE 8276 */

385 1    COMMAND$ADDRESS=00H; /* RESET THE 8276 */
386 1    PARAMETER$ADDRESS=4FH; /* NORMAL ROWS, 80 CHARACTER/ROW */
387 1    PARAMETER$ADDRESS=58H; /* 2 ROW COUNTS PER VERTICAL RETRACE
25 ROWS PER FRAME */
388 1    PARAMETER$ADDRESS=89H; /* LINE 9 IS THE UNDERLINE POSITION
10 LINES PER ROW */
389 1    PARAMETER$ADDRESS=0F9H; /* OFFSET LINE COUNTER, NON-TRANSPARENT FIELD ATTRIBUTE

```

PL/M-51 COMPILER CRTCONTROLLER

```

NON-BLINKING UNDERLINE CURSER, 20 CHARACTER COUNTS PER
HORIZONTAL RETRACE */
390 1      TEMP=COMMAND$ADDRESS;

391 1      CURSER$COLUMN=00H;
392 1      CURSER$ROW=00H;
393 1      CURSER$COL=00H;
394 1      CALL LOAD$CURSER;
395 1      TEMP=COMMAND$ADDRESS;

396 1      COMMAND$ADDRESS=0E0H;          /* PRESET 8276 COUNTERS */
397 1      TEMP=COMMAND$ADDRESS;

398 1      COMMAND$ADDRESS=23H;          /* START DISPLAY */
399 1      COMMAND$ADDRESS=0A0H;          /* ENABLE INTERRUPTS */
400 1      TEMP=COMMAND$ADDRESS;

      /* SET UP INTERRUPTS AND PRIORITIES */

      $IF SW1
401 1      IP=10H;                      /* SERIAL PORT HAS HIGHEST PRIORITY */
402 1      IE=85H;                      /* ENABLE 8051 EXTERNAL INTERRUPTS */
      $ENDIF

      $IF SW2
      IP=10H;                      /* SERIAL PORT HAS HIGHEST PRIORITY */
      IE=87H;                      /* ENABLE TIMER0 INTERRUPT */
      TMOD=05H;                    /* TIMER 0 =EVENT COUNTER */
      TLO=0FFH;
      TH0=0FFH;                    /* INITIALIZE COUNTER TO FFFFH */
      TR0=1;
      $ENDIF

      /* PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A
      SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT
      WILL BE TRANSMITTED TO THE HOST COMPUTER.*/

403 1      SCANNER:
      EA=1;
404 1      DATA$TERMINAL$READY=0;
405 1      IF CURSER$COUNT=1FH THEN    /* PROGRAMMABLE CURSER BLINK */
406 2      DO;
407 2          CURSER$ON=NOT CURSER$ON;
408 2          CURSER$COUNT=00;
409 2          IF CURSER$ON=0 THEN
410 2              CURSER$COL=7FH;
411 2          CALL LOAD$CURSER;
412 2      END;
413 1      IF LLC<>LOCAL$LINE THEN    /* IF LOCAL/LINE HAS CHANGED STATUS */
414 2      DO;
415 2          IF LOCAL$LINE=0 THEN
416 3          DO;
417 3              ENSP=0;
418 3              ES=0;
419 3          END;
420 2          ELSE
      CALL CHECK$BAUD$RATE;
421 2          LLC=LOCAL$LINE;
422 2      END;
      $IF SW1
423 2      DO WHILE SCAN$INT=0;        /* WAIT UNTIL VERTICAL RETRACE BEFORE */
424 2          IF SERIAL$INT=1 THEN    /* SCANNING THE KEYBOARD */
425 2              CALL DECIPHER;
426 2      END;

```

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT

427 1 CALL READER;
428 1 IF VALID$KEY =1 AND SAME=1 AND (LAST$SHIFT$KEY=SHIFT$KEY) AND
    (LAST$CAP$LOCK=CAP$LOCK) AND (LAST$CONTROL$KEY=CONTROL$KEY) THEN
429 1 CALL AUTO$REPEAT;
430 1 ELSE
    DO;
431 2 IF KEY0=0 AND SAME=0 THEN
432 3 DO;
433 3 TEMP =0;
434 3 K=0;
435 4 DO WHILE LAST$KEY (K)=0;
436 4 K=K+1;
437 4 END;
438 3 TEMP=LAST$KEY (K);
439 4 DO I=(K+1) TO 7;
440 4 TEMP=TEMP+LAST$KEY (I);
441 4 END;
442 3 IF TEMP=LAST$KEY (K) THEN
443 4 DO;
444 4 J=0;
445 5 DO WHILE (TEMP AND 01H)=0;
446 5 TEMP=SHR (TEMP,1);
447 5 J=J+1;
448 5 END;
449 4 IF TEMP >1 THEN
450 5 DO;
451 5 VALID$KEY=0;
452 5 NEWSKEY=0;
453 5 END;
454 4 ELSE
    DO;
455 5 IF CONTROL$KEY=0 THEN
456 5 ASCII$KEY=(LOW$SCAN(K).KEY(J)) AND 1FH;
457 5 ELSE
    DO;
458 6 IF SHIFT$KEY=0 THEN
459 6 ASCII$KEY=LOW$SCAN(K+08H).KEY(J);
460 6 ELSE
    DO;
461 7 ASCII$KEY=LOW$SCAN(K).KEY(J);
462 7 IF (CAP$LOCK=0) AND (ASCII$KEY>60H) AND (ASCII$KEY<7BH) THEN
463 7 ASCII$KEY=ASCII$KEY-20H;
464 7 IF LLC=0 THEN
465 8 DO;
466 8 IF ASCII$KEY=1BH THEN
467 8 ESC$SEQ=1;
468 8 ELSE
    ESC$SEQ=0;
469 8 END;
470 7 END;
471 6 END;
472 5 LAST$SHIFT$KEY=SHIFT$KEY;
473 5 LAST$CAP$LOCK=CAP$LOCK;
474 5 LAST$CONTROL$KEY=CONTROL$KEY;
475 5 VALID$KEY=1;
476 5 NEWSKEY=1;
477 5 END;
478 4 END;
479 3 ELSE
    DO;
480 4 VALID$KEY=0;
481 4 NEWSKEY=0;
482 4 END;
483 3 END;
484 2 END;

$ENDIF

```




PL/M-51 COMPILER CRICONTROLLER

```

$EJECT

$IF SW2
IF SERIAL$INT=1 THEN
  CALL DECIPHER;
IF KBDINT =1 THEN
DO;
  IF ERROR =0 THEN
  DO;
    ASCII$KEY=LST$KEY(1);
    NEWS$KEY=1;
    CALL AUTO$REPEAT;
    KBDINT=0;
  END;
  ERROR=0;
  KBDINT=0;
END;
$ENDIF

485 1  GOTO SCANNER;
486 1  END;
```

MODULE INFORMATION:	(STATIC+OVERLAYABLE)
CODE SIZE	= 08E6H 2278D
CONSTANT SIZE	= 0080H 128D
DIRECT VARIABLE SIZE	= 2DH+00H 45D+ 0D
INDIRECT VARIABLE SIZE	= 00H+00H 0D+ 0D
BIT SIZE	= 10H+00H 16D+ 0D
BIT-ADDRESSABLE SIZE	= 00H+00H 0D+ 0D
AUXILIARY VARIABLE SIZE	= 0000H 0D
MAXIMUM STACK SIZE	= 000CH 12D
REGISTER-BANK(S) USED:	0
1056 LINES READ	
0 PROGRAM ERROR(S)	
END OF PL/M-51 COMPILATION	

MCS-51 MACRO ASSEMBLER CNTASM

ISIS-11 MCS-51 MACRO ASSEMBLER V2.1
 OBJECT MODULE PLACED IN :F1:CNTASM.OBJ
 ASSEMBLER INVOKED BY: ASM51 :F1:CN1ASM.SRC

```

LVC OBJ      LINE SOURCE
              1
              2
              3
              4 ;
              5 ;
              6
              7 PUBLIC BLANK
              8 PUBLIC BLINE
              9 PUBLIC FILL
             10 EXTRN DATA (LINE0,MASTER,POINT,SERIAL,FIFO,CURSER,COUNT,L)
             11 EXTRN BIT (SERINT,ESCSEQ,THNINT,SCAN)
             12
             13
             14 CSEG AT(03H)
0003 8020      15 SJMP VENT ;RESET RASTER TO LINE0 AND SCAN KEYBOARD
             16
             17 ; EXTRN CODE (DETACH)
             18 ; CSEG AT(0BH)
             19 ; LJMP DETACH ;NEEDED IF DECODED KEYBOARD IS USED
             20
             21 CSEG AT(013H)
0013 802A      22 SJMP BUFFER ;FILL 8276 ROW BUFFER
             23
             24 CSEG AT(023H)
0023 802D      25 SJMP SERBUF ;STICK SERIAL INFORMATION INTO THE FIFO
             26
             27 CSEG
             28
0025 0000      29 VENT: PUSH PSH ;PUSH REG USED BY PLM51
0027 00E0      30 PUSH ACC
0029 0000      31 PUSH 00H
002d 850000 F 32 MOV MASTER,LINE0 ;REINITIALIZE MASTER TO LINE0
002e 850000 F 33 MOV MASTER+1,LINE0+1
0031 7801      34 MOV R0,#01H ;CLR 8276 INTERRUPT FLAG
0033 E2        35 MOVX A,4RH
0034 0500 F 36 INC COUNT ;INCR CURSER COUNT REGISTER
0036 0200 F 37 SETB SCAN ;FOR DEBOUNCE ROUTINE
0038 0000      38 POP 00H ;POP REGISTERS
003A 00E0      39 POP ACC
003C 0000      40 POP PSH
003E 32        41 RETI
             42
             43
003F 0000      44 BUFFER: PUSH PSH ;PUSH ALL REG USED BY PLM51 CODE
0041 00E0      45 PUSH ACC
0043 00B2      46 PUSH UPL
0045 00B3      47 PUSH UPH
0047 11FA      48 ACALL DMA ;FILL 8276 ROW BUFFER
0049 00B3      49 POP UPH ;POP REGISTERS
004B 00B2      50 POP UPL
             51
004D 00E0      51 POP ACC
004F 0000      52 POP PSH
0051 32        53 RETI
             54
             55 +1 SEJECT

```

MCS-51 MACRO ASSEMBLER CRTASM

```

LUC  OBJ      LINE  SOURCE
                                56
0052 309904    57  SERBUF: JNB  099H,CVER  ;IF TRANSMIT BIT NOT SET THEN CHECK RECEIVE
0055 0299      58      CLR  099H      ;CLR TRANSMISSION INTERRUPT FLAG
0057 0200      59      SETB  IRRINT    ;SET TRANS INT FOR PLM51 STATUS CHECK
0059 209828    60  OVER: JB   98H,GCBACK ;IF RI NOT SET GCBACK
005C 0001      61      PUSH  01H
005E A999      62      MOV   R1,80H      ;READ SBUF
0060 0298      63      CLR  098H      ;CLEAR RI BIT
0062 0000      64      PUSH  PSW      ;PUSH REGISTERS USED BY PLM51
0064 0000      65      PUSH  ACC
0066 0000      66      PUSH  00H
0068 0200      67      CLR  ESCSEQ    ;CLR ESC SEQUENCE FLAG
006A 7400      68      MOV   A,#SERIAL ;GET SERIAL FIFO RAM START LOCATION
006C 2500      69      ADD   A,#IFC     ;AND FIND HOW FAR INTO THE FIFO WE ARE
006E F8        70      MOV   R0,A      ;PUT IT INTO R0
006F E9        71      MOV   A,R1
0070 02E7      72      CLR  0E7H      ;CLR BIT 7 OF ACC
0072 F6        73      MOV   0R0,A      ;PUT DATA IN FIFO
0073 B41B02    74      CJNE  A,#10H,OVER1 ;IF DATA IS NOT A ESC KEY THEN GO OVER
0076 0200      75      SETB  ESCSEQ    ;SET ESC SEQUENCE FLAG
0078 0500      76  OVER1: INC   FIFC     ;MOV FIFC TO NEXT LOCATION
007A 0200      77      SETB  SERINT    ;SET SERIAL INT BIT FOR PLM51 STATUS CHECK
007C 0000      78      POP   00H      ;POP REGISTERS
007E 0000      79      POP   ACC
0080 0000      80      POP   PSW
0082 0001      81      POP   01H
0084 32        82  GCBACK: RETI
                                83
0085 0000      84  BLANK: PUSH  PSW      ;PUSH REG USED BY PLM51
0087 0000      85      PUSH  ACC
0089 0002      86      PUSH  DPL
008B 0003      87      PUSH  DPH
008D 0000      88      PUSH  00H
008F 850002    89      MOV   DPL,LINE0+1 ;GET LINE0 INFO
0092 850003    90      MOV   DPH,LINE0  ;AND PUT IT INTO DPH
0095 7850      91      MOV   R0,#50H   ;NUMBER OF CHARACTERS IN A LINE
0097 7400      92  NOTYET: MOV   A,#20H   ;ASCII SPACE CHARACTER
0099 F0        93      MOVX  0DPTR,A     ;MOV TO DISPLAY RAM
009A A3        94      INC   DPTR      ;INCR TO NEXT DISPLAY RAM LOCATION
009B 08FA      95      UJNZ  R0,NOTYET  ;IF ALL 50H LOCATIONS ARE NOT FILLED
                                96      ;GO DO MORE
009D 0000      97      POP   00H      ;POP REGISTERS
009F 0003      98      POP   DPH
00A1 0002      99      POP   DPL
00A3 0000     100      POP   ACC
00A5 0000     101      POP   PSW
00A7 22       102      RET
                                103 +1 $EJECT

```

MCS-51 MACRO ASSEMBLER CRTASM

LUC OBJ LINE SOURCE

```

104
00A8 C000      105 BLIN:  PUSH PSW          ;PUSH REGISTERS USED BY PLM51
00AA C0E0      106        PUSH ACC
00AC C082      107        PUSH LPL
00AE C083      108        PUSH UPH
00B0 C000      109        PUSH UCH
00B2 850083 F  110        MOV UPH,PC1N1      ;GET CURRENT DISPLAY RAM LOCATION
00B5 850082 F  111        MOV UPL,PC1N1+1
00B8 438310    112        URL UPH,#10H      ;SET BIT 15 FOR RAM ADDRESS DECODING
00BB 8B00      113        MOV R0,CURSEN ;GET CURSER COLUMN INFO TO TELL HOW
                                ;FAR INTO THE ROW YOU ARE
                                ;ASCII SPACE CHARACTER
00BD 7420      114 CONT1:  MOV A,#20H      ;ASCII SPACE CHARACTER
00BF F0        115        MOVX @DPTR,A      ;MOVE TO DISPLAY RAM
00C0 A3        116        INC DPTR      ;INCH TO NEXT DISPLAY RAM LOCATION
00C1 08        117        INC R0
00C2 8820F8    118        CJNE R0,#50H,CONT1 ;IF NOT AT THE END OF THE LINE
                                ;CONTINUE
                                ;POP REGISTERS
00C5 C000      119        POP UCH
00C7 0083      120        POP UPH
00C9 0082      121        POP UPL
00CB 00E0      122        POP ACC
00CD C000      123        POP PSW
00CF 22        124        RET
                                ;PUSH REGISTERS USED BY PLM51
00D0 C000      125        PUSH PSW
00D2 C0E0      126        PUSH ACC
00D4 C082      127        PUSH UPL
00D6 C083      128        PUSH UPH
00D8 C000      129        PUSH UCH
00DA C3        130        CLR C
00DB 850083 F  131        MOV UPH,L      ;GET BEGINNING OF LINE RAM LOCATION
00DE 850082 F  132        MOV UPL,L+1    ;CALCULATED BY PLM51
00E1 438310    133        URL UPH,#10H    ;SET BIT 15 FOR DISPLAY RAM ADDRESS DECODE
00E4 784F      134        MOV R0,#4FH    ;SET UP COUNTER FOR 50H LOCATIONS
00E6 A3        135        INC DPTR      ;GO PAST THE 0F1H
00E7 7420      136 CONT2:  MOV A,#20H      ;ASCII SPACE CHARACTER
00E9 F0        137        MOVX @DPTR,A    ;MOVE TO DISPLAY RAM
00EA A3        138        INC DPTR      ;INCH TO NEXT DISPLAY RAM LOCATION
00EB 0BF A     139        DJNZ R0,CUNT2 ;IF ALL 79 LOCATIONS HAVE NOT BEEN FILLED
                                ;THEN CONTINUE
                                ;POP REGISTERS
00ED 0000      140        POP UCH
00EF 0083      141        POP UPH
00F1 0082      142        POP UPL
00F3 00E0      143        POP ACC
00F5 0000      144        POP PSW
00F7 22        145        RET
                                ;
150
151
152 +1 SEJECT

```

MCS-51 MACRO ASSEMBLER CRTASM

```

LUC  CPJ      LINE  SOURCE
                                153
                                154 ;+++++
                                155 ;THIS ROUTINE MOVES DISPLAY RAM DATA TO MON BUFFER OF 8276
                                156 ;+++++
                                157
00F0 21dB      158 DDONE: AJMP    DDONE
                                159
00FA 850083 F  160 DMA:  MOV     DPM,MASTER      ;LOAD XFER POINTER HIGH BYTE
00FD 050082 F  161      MOV     CPL,MASTER+1      ;LOAD XFER POINTER LOW BYTE
0100 E0        162      MOVX    A,CDPTM
0101 A3        163      INC     DPTM
0102 2003F3    164      JB      VBST,DDONE      ;IF IN11 HIGH, THEN DMA IS OVER
0103 E0        165      MOVX    A,CDPTM
0106 A3        166      INC     DPTM
0107 E0        167      MOVX    A,CDPTM
0108 A3        168      INC     DPTM
0109 E0        169      MOVX    A,CDPTM
010A A3        170      INC     DPTM
010B E0        171      MOVX    A,CDPTM
010C A3        172      INC     DPTM
010D E0        173      MOVX    A,CDPTM
010E A3        174      INC     DPTM
010F E0        175      MOVX    A,CDPTM
0110 A3        176      INC     DPTM
0111 E0        177      MOVX    A,CDPTM
0112 A3        178      INC     DPTM
0113 E0        179      MOVX    A,CDPTM
0114 A3        180      INC     DPTM
0115 E0        181 TEN:  MOVX    A,CDPTM
0116 A3        182      INC     DPTM
0117 E0        183      MOVX    A,CDPTM
0118 A3        184      INC     DPTM
0119 E0        185      MOVX    A,CDPTM
011A A3        186      INC     DPTM
011B E0        187      MOVX    A,CDPTM
011C A3        188      INC     DPTM
011D E0        189      MOVX    A,CDPTM
011E A3        190      INC     DPTM
011F E0        191      MOVX    A,CDPTM
0120 A3        192      INC     DPTM
0121 E0        193      MOVX    A,CDPTM
0122 A3        194      INC     DPTM
0123 E0        195      MOVX    A,CDPTM
0124 A3        196      INC     DPTM
0125 E0        197      MOVX    A,CDPTM
0126 A3        198      INC     DPTM
0127 E0        199      MOVX    A,CDPTM
0128 A3        200      INC     DPTM
0129 E0        201 TWENTY: MOVX   A,CDPTM
012A A3        202      INC     DPTM
012B E0        203      MOVX    A,CDPTM
012C A3        204      INC     DPTM
012D E0        205      MOVX    A,CDPTM
012E A3        206      INC     DPTM
012F E0        207      MOVX    A,CDPTM

```

MCS-51 MACRO ASSEMBLER CRTASK

LOC	OBJ	LINE	SOURCE
0130	A3	208	INC UPTR
0131	E0	209	MOVX A,ADPTR
0132	A3	210	INC UPTR
0133	E0	211	MOVX A,ADPTR
0134	A3	212	INC UPTR
0135	E0	213	MOVX A,ADPTR
0136	A3	214	INC UPTR
0137	E0	215	MOVX A,ADPTR
0138	A3	216	INC UPTR
0139	E0	217	MOVX A,ADPTR
013A	A3	218	INC UPTR
013B	E0	219	MOVX A,ADPTR
013C	A3	220	INC UPTR
013D	E0	221	THIRTY: MOVX A,ADPTR
013E	A3	222	INC UPTR
013F	E0	223	MOVX A,ADPTR
0140	A3	224	INC UPTR
0141	E0	225	MOVX A,ADPTR
0142	A3	226	INC UPTR
0143	E0	227	MOVX A,ADPTR
0144	A3	228	INC UPTR
0145	E0	229	MOVX A,ADPTR
0146	A3	230	INC UPTR
0147	E0	231	MOVX A,ADPTR
0148	A3	232	INC UPTR
0149	E0	233	MOVX A,ADPTR
014A	A3	234	INC UPTR
014B	E0	235	MOVX A,ADPTR
014C	A3	236	INC UPTR
014D	E0	237	MOVX A,ADPTR
014E	A3	238	INC UPTR
014F	E0	239	MOVX A,ADPTR
0150	A3	240	INC UPTR
0151	E0	241	FORTY: MOVX A,ADPTR
0152	A3	242	INC UPTR
0153	E0	243	MOVX A,ADPTR
0154	A3	244	INC UPTR
0155	E0	245	MOVX A,ADPTR
0156	A3	246	INC UPTR
0157	E0	247	MOVX A,ADPTR
0158	A3	248	INC UPTR
0159	E0	249	MOVX A,ADPTR
015A	A3	250	INC UPTR
015B	E0	251	MOVX A,ADPTR
015C	A3	252	INC UPTR
015D	E0	253	MOVX A,ADPTR
015E	A3	254	INC UPTR
015F	E0	255	MOVX A,ADPTR
0160	A3	256	INC UPTR
0161	E0	257	MOVX A,ADPTR
0162	A3	258	INC UPTR
0163	E0	259	MOVX A,ADPTR
0164	A3	260	INC UPTR
0165	E0	261	FIFTY: MOVX A,ADPTR
0166	A3	262	INC UPTR

MCS-51 PALRU ASSEMBLER CNTASM

LOC	UPJ	LINE	SOURCE
0167	E0	263	MOVX A,CDPTM
0168	A3	264	INC UPTR
0169	E0	265	MOVX A,CDPTM
016A	A3	266	INC UPTR
016B	E0	267	MOVX A,CDPTM
016C	A3	268	INC UPTR
016D	E0	269	MOVX A,CDPTM
016E	A3	270	INC UPTR
016F	E0	271	MOVX A,CDPTM
0170	A3	272	INC UPTR
0171	E0	273	MOVX A,CDPTM
0172	A3	274	INC UPTR
0173	E0	275	MOVX A,CDPTM
0174	A3	276	INC UPTR
0175	E0	277	MOVX A,CDPTM
0176	A3	278	INC UPTR
0177	E0	279	MOVX A,CDPTM
0178	A3	280	INC UPTR
0179	E0	281	SIXTY: MOVX A,CDPTM
017A	A3	282	INC UPTR
017B	E0	283	MOVX A,CDPTM
017C	A3	284	INC UPTR
017D	E0	285	MOVX A,CDPTM
017E	A3	286	INC UPTR
017F	E0	287	MOVX A,CDPTM
0180	A3	288	INC UPTR
0181	E0	289	MOVX A,CDPTM
0182	A3	290	INC UPTR
0183	E0	291	MOVX A,CDPTM
0184	A3	292	INC UPTR
0185	E0	293	MOVX A,CDPTM
0186	A3	294	INC UPTR
0187	E0	295	MOVX A,CDPTM
0188	A3	296	INC UPTR
0189	E0	297	MOVX A,CDPTM
018A	A3	298	INC UPTR
018B	E0	299	MOVX A,CDPTM
018C	A3	300	INC UPTR
018D	E0	301	SEVENTY: MOVX A,CDPTM
018E	A3	302	INC UPTR
018F	E0	303	MOVX A,CDPTM
0190	A3	304	INC UPTR
0191	E0	305	MOVX A,CDPTM
0192	A3	306	INC UPTR
0193	E0	307	MOVX A,CDPTM
0194	A3	308	INC UPTR
0195	E0	309	MOVX A,CDPTM
0196	A3	310	INC UPTR
0197	E0	311	MOVX A,CDPTM
0198	A3	312	INC UPTR
0199	E0	313	MOVX A,CDPTM
019A	A3	314	INC UPTR
019B	E0	315	MOVX A,CDPTM
019C	A3	316	INC UPTR
019D	E0	317	MOVX A,CDPTM

MCS-51 MACRO ASSEMBLER CRTASM

LOC	OBJ	LINE	SOURCE
019E	A3	310	INC DPH
019F	E0	311	MOVX A, @DPTR
01A0	A3	320	INC DPH
01A1	E0	321	EIGHTY: MOVX A, @DPTR
01A2	A3	322	INC DPH
		323	
01A3	E503	324	CHECK: MOV A, DPH
01A5	041F0C	325	CJNE A, #1FH, DONE
01A6	E502	326	MOV A, CPL
01AA	040007	327	CJNE A, #000H, DONE
01AD	750018	328	MOV RASTER, #18h
01B0	750000	329	MOV RASTER+1, #00h
01B3	22	330	RET
		331	
01B4	050300	332	DONE: MOV RASTER, DPH
01B7	050200	333	MOV RASTER+1, DPH
01BA	22	334	RET
		335	
01BB	03	336	UMADNE: CLR C
01B0	E502	337	MOV A, DPH
01BE	244F	338	ADD A, #79D
01C0	F502	339	MOV DPH, A
01C2	500F	340	JNC CHECK
01C4	0503	341	INC DPH
01C6	000B	342	SJMP CHECK
		343	
		344	
		345	END

ADD 79 TO BUFFER POINTER
TO GET TO NEXT DISPLAY LINE
IN THE DISPLAY MEMORY



MCS-51 MACRO ASSEMBLER CNTASM

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
ACC. . . .	D ADDR	00E0H	A
BLANK. . .	C ADDR	0085H	A PUB
BLINE. . .	C ADDR	00A8H	A PUB
BUFFER. . .	C ADDR	003FH	A
CHECK. . .	C ADDR	01A3H	A
CUN11. . .	C ADDR	00B6H	A
CUNT2. . .	C ADDR	00E7H	A
COUNT. . .	D ADDR	----	EXT
CURSEK. . .	D ADDR	----	EXT
DDONE. . .	C ADDR	00F8H	A
DMA. . . .	C ADDR	00FAH	A
DMADNE. . .	C ADDR	01B8H	A
DONE. . . .	C ADDR	0184H	A
DPH. . . .	D ADDR	00E3H	A
DPL. . . .	D ADDR	00E2H	A
EIGHTY. . .	C ADDR	01A1H	A
ESCSEW. . .	B ADDR	----	EXT
FIFU. . . .	D ADDR	----	EXT
FIFTY. . . .	C ADDR	0165H	A
FILL. . . .	C ADDR	00C0H	A PUB
FURTY. . .	C ADDR	0151H	A
GUBACK. . .	C ADDR	0084H	A
L.	D ADDR	----	EXT
LINE0. . .	D ADDR	----	EXT
NUTYET. . .	C ADDR	0097H	A
OVER. . . .	C ADDR	0059H	A
OVER1. . .	C ADDR	0078H	A
POINT. . .	D ADDR	----	EXT
PSW. . . .	D ADDR	00C0H	A
RASIER. . .	D ADDR	----	EXT
SBUF. . . .	D ADDR	0099H	A
SCAN. . . .	B ADDR	----	EXT
SERBUF. . .	C ADDR	0052H	A
SERIAL. . .	D ADDR	----	EXT
SERINT. . .	B ADDR	----	EXT
SEVENTY. . .	C ADDR	0180H	A
SIXTY. . . .	C ADDR	0179H	A
TEN. . . .	C ADDR	0115H	A
THIRTY. . .	C ADDR	0130H	A
THNINI. . .	B ADDR	----	EXT
TWENTY. . .	C ADDR	0129H	A
VERI. . . .	C ADDR	0025H	A

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

MCS-51 MACRO ASSEMBLER KEYBD

ISIS-II MCS-51 MACRO ASSEMBLER v2.1
 OBJECT MODULE PLACED IN :F1:KEYBD.OBJ
 ASSEMBLER INVOKED BY: ASM51 :F1:KEYBD.SRC

```

LOC  OBJ  LINE      SOURCE
      1
      2
      3
      4      ;*****
      5      ;*****
      6      ;****
      7      ;****          SOFTWARE FOR READING AN UNDECODED          ****
      8      ;****          KEYBOARD          ****
      9      ;****          ****
     10      ;*****
     11      ;*****
     12      ;
     13      ;
     14      ;
     15      ;
     16      ;          THIS CONTAINS THE SOFTWARE NEEDED TO SCAN AN UNDECODED KEYBOARD
     17      ;          THIS PROGRAM MUST BE LINKED TO THE MAIN PROGRAMS TO FUNCTION
     18      ;
     19      ;
     20      ;          MEMORY MAP FOR READING KEY BOARD (USING MOVX)
     21      ;
     22      ;          ADDRESS FOR KEY BOARD 10FFH TO 17FFH
     23      ;
     24      ;
     25      ;
     26      ;
     27      ;          PUBLIC READEN
     28      ;          EXTRN DATA (LSIKEY)
     29      ;          EXTRN BIT (KEY0,SAME)
     30      ;
     31      ;*****
     32      ;*
     33      ;*          "READEN"ROUTINE"          *
     34      ;*
     35      ;*****
     36 +1  OBJECT
  
```

MUS-51 MACRO ASSEMBLER KEYBD

LOC OBJ LINE SOURCE

```

3/
38 UNDECODED_KEYBOARD SEGMENT CODE
---- 39 KSEG UNDECODED_KEYBOARD
40
41
42
43 HEADER: PUSH PSW ;PUSH REG USED BY PLM51
44 PUSH ACC
45 PUSH UPL
46 PUSH UPM
47 PUSH U0M
48 PUSH U1M
49 PUSH U2M
50 PUSH U3M
51 MOV UPTR,#10FFH ;INITIALIZE UPTR TO KEYBOARD
52 ;ADDRESS
0013 7900 53 MOV R1,#00H ;CLR ZERO COUNTER
0015 7800 F 54 MOV R0,#LSIKEY ;GET KEYBOARD RAM POINTER
0017 7808 55 MOV R3,#08H ;INITIALIZE LOOP COUNTER
0019 0200 F 56 CLR KEY0 ;INITIALIZE PLM51 STATUS BITS
001B 0200 F 57 SETB SAME
001D 0602 58 MORE: MOV 02H,R0 ;MOV LAST KEYBOARD SCAN TO 02H
001F E4 59 CLR A
0020 93 60 MOVC A,A+DPTH ;SCAN KEYBOARD
0021 F4 61 CPL A ;INVERT
0022 0005 62 JZ ZERO ;IF SCAN WAS ZERO GO INCREMENT ZERO COUNTER
0024 05024 63 CJNE A,02H,NTSAME ;COMPARE WITH LAST SCAN IF NOT THE SAME
64 ;THEN CLR SAME BIT AND WRITE NEW INFORMATION
65 ;TO RAM
0027 0005 66 SJMP EQUAL ;IF EQUAL JMP OVER INCH OF ZERO COUNTER
0029 0501 67 ZERO: INC 01H ;INCR ZERO COUNTER
002B 050210 68 CJNE A,02H,NTSAME
002E 08 69 EQUAL: INC R0 ;STEP TO NEXT SCAN RAM LOCATION
002F 0503 70 INC UPM ;NEXT KEYBOARD ADDRESS
0031 08EA 71 DJNZ R3,MORE ;IF LOOP COUNTER NOT 0, SCAN AGAIN
0033 090804 72 CJNE R1,#08H,BACK ;CHECK TO SEE IF ALL 8 SCANS WHERE 0
0035 0200 F 73 SETB KEY0 ;IF YES SET KEY0 BIT
003B 0200 F 74 CLR SAME
003A 0003 75 BACK: POP 03H ;POP REGISTERS
003C 0002 76 POP 02H
003E 0001 77 POP 01H
0040 0000 78 POP 00H
0042 0003 79 POP UPM
0044 0002 80 POP UPL
0046 0000 81 POP ACC
0048 0000 82 POP PSW
004A 02 83 RET
84
004B F6 85 NTSAME: MOV R0,A ;IF SCAN WAS NOT THE SAME THEN PUT NEW
86 ;SCAN INFO INTO RAM
004C 0200 F 87 CLR SAME BIT
004E 000E 88 SJMP EQUAL ;GO DO MORE
89
90
91 END

```



MCS-51 MACRO ASSEMBLER KEYBD

SYMBOL TABLE LISTING

N A M E	T Y P E	V A L U E	A T T R I B U T E S
ALC.	D AUCH	00E6H	A
BACK	C AUCH	003AH	R SEG=UNDECODED_KEYBOARD
DPM.	D AUCH	0083H	A
DPL.	D AUCH	0082H	A
EQUAL.	C AUCH	002EH	R SEG=UNDECODED_KEYBOARD
KEYU	E AUCH	----	EXT
LSTKEY	D AUCH	----	EXT
MURE	C AUCH	0016H	R SEG=UNDECODED_KEYBOARD
NISAME	C AUCH	0046H	R SEG=UNDECODED_KEYBOARD
PSK.	C AUCH	00D0H	A
READER	C AUCH	00D0H	R PUB SEG=UNDECODED_KEYBOARD
SAME	E AUCH	----	EXT
UNDECODED_KEYBOARD	C SEG	0050H	REL=LN11
ZERU	C AUCH	0029H	R SEG=UNDECODED_KEYBOARD

REGISTER BANK(S) USED: 0
ASSEMBLY COMPLETE, NO ERRORS FOUND



MCS-51 MACRO ASSEMBLER DECODE

```

ISIS=11 MCS-51 MACRO ASSEMBLER V2.1
OBJECT MODULE PLACED IN :F1:DECODE.OBJ
ASSEMBLER INVOKED BY:  ASM51 :F1:UICODE.SRC

```

```

LOC  OBJ  LINE      SOURCE
      1
      2
      3
      4      ;*****
      5      ;*****
      6      ;****
      7      ;****      SOFTWARE FOR DECODED KEYBOARD      ****
      8      ;****
      9      ;*****
     10      ;*****
     11      ;
     12      ;
     13      ;
     14      ;
     15      PUBLIC DETACH
     16      EXTRN DATA (LSKEY)
     17      EXTRN INT (KBDINT)
     18
     19
     20      ;
     21      ;*****
     22      ;*
     23      ;*      "DECODE" INTERRUPT ROUTINE FOR DECODED KEYBOARD      *
     24      ;*
     25      ;*****
     26 +1  $EJECT

```

PLS-51 MACRO ASSEMBLER DECODE

```

LUC  OBJ      LINE  SOURCE
                                27
                                28  DECODED_KEYBOARD SEGMENT CODE
                                29  KSEG DECODEC_KEYBOARD
                                30
-----
0000 0000      31  DETACH: PUSH    PSW          ;PUSH REGISTERS
0002 0002      32          PUSH    DPL          ;USED BY PLM51
0004 0003      33          PUSH    DPH
0006 0000      34          PUSH    ACL
0008 0000FFFF  35          MOV     DFI2,#00FFFF ;ADDRESS FOR KEYBOARD
000B 04      36          CLK     A
000C 93      37          MOVC    A,A+DPTH      ;FETCH ASCII BYTE
000D F500      38          MOV     LSIKEY+1,A    ;MOV TO MEMORY TO BE READ BY PLM51
000F 0200      39          SETB    KBUINT      ;LET PLM51 KNOW THERE IS A BYTE
0011 7500FF    40          MOV     IPU,#00FF    ;SET COUNTER TO FFFF SO INTERRUPT
0014 7500FF    41          MOV     TL0,#00FF    ;ON THE NEXT FALLING EDGE OF T0
0017 0000      42          POP     ACC
0019 0003      43          POP     DPH          ;POP REGISTERS
001B 0002      44          POP     DPL
001D 0000      45          POP     PSW
001F 32      46          RETI
                                47
                                48
                                49
                                50
                                51  END

```

CS-51 MACRO ASSEMBLER DECODE

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
_CC.	D ADDR	00E0H	A
_DECODED_KEYBOARD	C SEG	0020H	REL=LNII
_DETACH.	C ADDR	0000H	R PUB SEG=DECODED_KEYBOARD
_PM.	D ADDR	0083H	A
_PL.	D ADDR	0082H	A
_BDINI.	B ADDR	----	EXT
_STKEY.	D ADDR	----	EXT
_SW.	D ADDR	00D0H	A
_MO.	D ADDR	008CH	A
_LO.	D ADDR	008AH	A

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND



```
MCS-51 MACRO ASSEMBLER DETACH

THIS IS MCS-51 MACRO ASSEMBLER V2.1
OBJECT MODULE PLACED IN :F1:DETACH.OBJ
ASSEMBLER INVOKED BY: ASM51 :F1:DETACH.SRC

LUC OBJ LINE SOURCE
1
2
3
4 ;*****
5 ;*****
6 ;****
7 ;**** SOFTWARE FOR A SERIAL OR DETACHABLE ****
8 ;**** KEYBOARD ****
9 ;****
10 ;*****
11 ;*****
12 ;
13 ;
14 ;
15 ; THIS CONTAINS THE SOFTWARE NEEDED TO PERFORM A SOFTWARE SERIAL
16 ; PORT FOR SERIAL KEYBOARDS AND DETACHABLE KEYBOARD. THIS PROGRAM MUST
17 ; BE LINKED TO THE MAIN PROGRAMS FOR USE.
18 ;
19 +1 $EJECT
```



MCS-51 MACRO ASSEMBLER DETACH

```

LUC OBJ      LINE      SOURCE
                20      ;
                21      ;
                22      ;
00B4          23      INPUT EQU TV
                24
                25
                26      PUBLIC DETACH
                27      EXTRN DATA (LSIKEY)
                28      EXTRN BIT (RCVPLG,SYNC,BYFIA)
                29      EXTRN BIT (KBDINI,ERRKON)
                30
                31      ;
                32      ;
                33      ;
                34
                35      ;      TIMER U LOAD VALUES FOR DIFFERENT BAUD RATES
                36      ;      USED WITH DETACHABLE KEYBOARDS
                37      ;
                38      ;
                39      ;
                40      ;      BAUD      START BIT DETECT      MESSAGE DETECT
                41      ;      110      0EFA2H      0DF45H
                42      ;      150      0F400H      0E800H
                43      ;
                44      ;
                45      ;
                46      ;
0000          47      START0      EQU      000H      ;LOW BYTE FOR 150 BAUD
00F4          48      START1      EQU      0F4H      ;HIGH BYTE FOR 150 BAUD
0000          49      MESSAGE0     EQU      000H      ;LOW BYTE FOR 150 BAUD
00E8          50      MESSAGE1     EQU      0E8H      ;HIGH BYTE FOR 150 BAUD
                51 +1 $EJECT

```


DETACH

MCS-51 MACRO ASSEMBLER DETACH

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
ACC	U ADDR	00E0H	A
BYFIN	B ADDR	----	EXT
DETACH	C ADDR	0000H	N PUB SEG=DETACHABLE_KEYBOARD
DETACHABLE_KEYBOARD	C SEG	0068H	NEL=UNIT
ERR	C ADDR	0052H	N SEG=DETACHABLE_KEYBOARD
ERRR	B ADDR	----	EXT
FINI	C ADDR	0045H	N SEG=DETACHABLE_KEYBOARD
INPUT	B ADDR	00B0H.4	A
KBDINT	B ADDR	----	EXT
LSTKEY	U ADDR	----	EXT
MESSAGE0	NLMB	0000H	A
MESSAGE1	NLMB	00E8H	A
NATBIT	C ADDR	002DH	N SEG=DETACHABLE_KEYBOARD
PSW	U ADDR	0000H	A
RCVFLG	B ADDR	----	EXT
RST	C ADDR	0054H	R SEG=DETACHABLE_KEYBOARD
START0	NLMB	0000H	A
START1	NLMB	00F4H	A
STOP	C ADDR	004AH	N SEG=DETACHABLE_KEYBOARD
SYNC	B ADDR	----	EXT
TU	B ADDR	00B0H.4	A
TH0	U ADDR	008CH	A
TLO	U ADDR	008AH	A
THOU	U ADDR	0089H	A
VALID	C ADDR	001AH	R SEG=DETACHABLE_KEYBOARD

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND



APPENDIX B REFERENCES

1. John Murray and George Alexy, *CRT Terminal Design Using The Intel 8275 and 8279*, Intel Application Note AP-32, Nov., 1977.
2. John Katausky, *A Low Cost CRT Terminal Using The 8275*, Intel Application Note AP-62, Nov., 1979.